Studies on the Space Exploration and the Sink Location under Incomplete Information towards Applications to Evacuation Planning

Yuya Higashikawa

Studies on the Space Exploration and the Sink Location under Incomplete Information towards Applications to Evacuation Planning

不完全情報下における空間探索及び施設配置に関する理論的研究

～ 避難計画への応用を目指して ～

In Partial Fulfillment
of the Requirement for the Degree of
Doctor of Engineering Department of Architecture and
Architectural Engineering

Kyoto University

Yuya Higashikawa
東川 雄哉

September, 2014

Acknowledgements

I would like to thank my dissertation supervisor, Professor Naoki Katoh for his support. I could get a number of helpful suggestions from his deep insight and inspiration. I would also like to express my gratitude to Professor Mordecai J. Golin for his patience, kindness and guidance. In addition, I would like to thank the other members of my dissertation committee, Professor Teruyuki Monnai and Professor Kiyoko Kanki for taking the time and effort to read my dissertation. I wish to thank Japan Society for the Promotion of Science (JSPS) for supporting my studies via Grant-in-Aid for JSPS Fellows. Finally, I am grateful to my family for their love and support.

Contents

Chapter 1

Introduction

Recently, big earthquakes frequently occur around the world, e.g., *the Great Hanshin-Awaji Earthquake* in 1995, *the Sichuan Earthquake* in 2008, and so on. Also, some big earthquakes triggers devastating tsunamis, e.g., *the Sumatra-Andaman Earthquake* in 2004 and *The Tohoku-Pacific Ocean Earthquake* in 2011. In such disasters, many people failed to evacuate and lost their lives due to severe attack by tsunamis. Moreover, not only earthquakes but also diverse disasters occur and cause serious damages in many countries. Therefore, from the viewpoint of disaster prevention, it has now become extremely important to establish effective *evacuation planning* systems against large scale disasters.

When a big earthquake occurs in an urban area, it is predicted that many buildings and underground shopping areas will be heavily damaged. And then, disaster victims in the damaged area will be required to search for exits (or signs leading to exits) and evacuate to safe area as quickly as possible. Also, it is seriously important to efficiently explore the inside of damaged areas in order to rescue human beings left there. Unfortunately, in the search of emergency, victims may be unfamiliar with the site, that is, they may not know the map of site nor the location of exits, and also, exits (or signs) may be less visible than usual due to the power failure or dust. On the other hand, even for the rescue workers who know the map of site, it would be difficult to know in advance where victims remain or which passages are so damaged that the rescue workers cannot path through. Therefore, it seems significant to consider how to explore a building or a city under incomplete information, and the obtained knowledge will implicitly reveal the geometric structure of building or city which makes the evacuation easier.

Another important problem arising from decision making in urban planning against large scale disasters such as tsunamis or earthquakes under incomplete

information is where to locate evacuation centers. Generally, the number of evacuees in an area may vary depending on the time, e.g., in an office area in a big city there are many people during the daytime on weekdays while there are much less people on weekends or during the night time. If we optimize the location of evacuation centers in a city assuming the fixed population distribution, the solution may not be robust for the unexpected scenarios. In order to obtain the robust location in this sense, it is required to take into account the uncertainty of the population distribution.

In this dissertation, considering the above two issues, we aim to formulate the following problems:

I. **Space exploration problems under incomplete information**

II. **Sink location problems in dynamic networks under incomplete information**

For each problem, we will propose an algorithm which solves the problem exactly, and also, show the efficiency of each proposed algorithm quantitatively. Eventually, from the above theoretical results, we aim to generalize about the quantitative evaluation for evacuation planning.

In the rest of this chapter, we explain these two problems and survey previous related works. Then, we summarize our contribution of this dissertation.

1.1 Space Exploration Problems under Incomplete Information

As mentioned above, the problem is motivated by the need to efficiently explore the inside of buildings or underground shopping areas damaged by large scale disasters in order to quickly rescue human beings left there. In the problem we consider, we are given an unknown environment called *the object space*, and a particular point in the object space called *the origin*. Note that an object space may be an undirected graph or the inside of a polygon in the Euclidean plane depending on the setting. Suppose that *the searchers* are initially given at the origin. At the beginning, they do not have any information about the object space and only have local information around the origin. They are also assumed to know that there exist several *targets* to be explored somewhere in the object space. As the exploration proceeds, the searchers gain more information of the environment and determine the next actions based on the

information obtained so far. We call such a problem setting *online*. In this dissertation, we define *the online space exploration problem* as the problem requiring that every target is visited by at least one searcher and all searchers return to the origin as quickly as possible in the online setting.

If an algorithm can process the input piece-by-piece in the order which the input is fed to the algorithm without having the whole input available at the beginning, it is called *an online algorithm* [44]. In contrast, *an offline optimal algorithm* is given the whole input at the beginning and solves the problem with an optimal solution. In our problem, the whole input corresponds to the complete map of the object space. Generally, the performance of an online algorithm is measured by *the competitive ratio* which is defined as follows. Let $\mathcal{X}$ denote a class of object spaces. When an online exploration algorithm ALG is used to explore an object space $X \in \mathcal{X}$, let $\mathsf{ALG}(X)$ denote the time required to complete the exploration of X by ALG. Also, let $\mathsf{OPT}(X)$ denote the time required to complete the exploration of X by an offline optimal algorithm. Then the competitive ratio of ALG is defined as follows:

$$cr_{\mathsf{ALG}} = \sup_{X \in \mathcal{X}} \frac{\mathsf{ALG}(X)}{\mathsf{OPT}(X)}.$$

In this dissertation, we will develop online algorithms for the online space exploration problems and analyze their competitive ratios. Since it is generally hard to explicitly show a competitive ratio, we will show their upper bound and lower bound. The upper bound of a competitive ratio ensures how inaccurate the developed algorithm is in the worst case comparing with an offline optimal algorithm, in other words, the worth of information for the class of object spaces. In the following, an algorithm of which the competitive ratio is at most α is called *α-competitive*. We also analyze the lower bound of competitive ratios for all online algorithms which work in the class of object spaces. Therefore, the obtained lower bound ensures how hard exploring such object spaces is.

In this dissertation, depending on the class of object spaces, we classify the problems into the two groups: *the online graph exploration problems* and *the online polygon exploration problems*. In the former problems, we treat undirected graphs as object spaces, e.g., cycles and trees. Given an input graph, we suppose that the searchers can move along edges and targets are all vertices of the graph. In the latter problems, we treat polygons in the Euclidean plane as object spaces, e.g., simple polygons and rectilinear simple polygons. Given an input polygon, we suppose that the searchers can freely

move in the inside of the polygon and targets are all vertices of the polygon.

1.1.1 Previous work

In the field of urban and architectural planning, several studies treated the space exploration under the incomplete information, e.g., studies on how ease of exploration is characterized in an object space [41, 42], and studies on the distinctive behavior of searchers in an emergency [46]. All these studies were based on the practical demonstration assuming the incomplete information, that is, participants (playing searchers or evacuees in the demonstration) were not given any map of site or forced to explore a site with blindfolds.

In the field of computer science, online graph exploration problems have been extensively studied so far. The theoretical study of online graph exploration problems was initiated by Deng et al. [14], who studied the problem of exploring not only all vertices but also all edges. The problem was further investigated by [1, 22, 38], and several variants were considered, e.g., [17] restricting the power of a searcher. Note that all these studies assumed that the exploration is done by a single searcher. When targets are all vertices of an input graph, the problem is also called the *online traveling salesman problem* (online TSP for short), e.g., in [29, 34] (notice that the terminology "online TSP" is also used to stand for a different problem, where demand requests from vertices are given online and the salesman needs to serve those requests by visiting vertices, e.g., in [4, 3, 8]).

In the case that the number of searcher is only one, several results are obtained for various classes of the graph. For planar graphs, Kalyanasundaram et al. [29] proposed a 16-competitive algorithm. Very recently, Megow et al. [34] generalized the upper bound analysis of [29] to the case for graphs of bounded genus g, and proposed a $16(1 + 2g)$-competitive algorithm.. Also for cycles, Miyazaki et al. [35] gave an optimal $(1 + \sqrt{3})/2$-competitive algorithm. Moreover for undirected graphs, in the context of the *advice complexity*, i.e., the total bit length of the additional information provided for the online algorithm during the exploration so that it achieve a competitive ratio of 1, Dobrev et al. [16] showed that any deterministic online exploration algorithm requires the advice complexity of $\Omega(n \log n)$, and gave a deterministic online exploration algorithm which uses $O(n)$ bits of the advice and achieves a constant competitive ratio.

In the case that the number of searcher is general p, several results are known for tree explorations. Fraigniaud et al. [24] gave an $O(p/\log p)$-competitive

algorithm and Dynia et al. [20] gave a lower bound of $\Omega(\log p / \log\log p)$ for the competitive ratio of any deterministic algorithm for trees. There are also results whose analyses are based on other parameters of trees. Dynia et al. [19] gave $O(H^{1-1/d} \cdot \min\{d, \log d \cdot H^{1/2d}\})$-competitive algorithm where H is the height and d is the density of the input tree. Dynia et al. [18] also studied the tree exploration problem with the different criteria, which is measured by the maximum tour length for all searchers instead of the running time, and showed an 8-competitive algorithm. Furthermore, regarding the time complexity, Brass et al. [9] gave $2n/p + O(H^{p-1})$-time algorithm for a tree which has the height of H and n vertices.

We should remark that the offline problem (i.e., computing p tours that covers all vertices with minimum cost) is NP-hard even for trees with $p = 2$, and several approximation algorithms are known [5, 36].

On the other hand, there are some studies which are related to the online space exploration problem in geometric regions (see survey paper [26]). Kalyanasundaram et al. [29] studied the case of a polygon with holes where all edges are required to traverse. They gave a 17-competitive algorithm for this case. Hoffmann et al. [27] studied the problem which asks to find a tour in a simple polygon such that every vertex is visible from some point on the tour, and gave a 26.5-competitive algorithm.

1.2 Sink Location Problems in Dynamic Networks under Incomplete Information

This problem is motivated by the need to determine the location of evacuation centers in a city to be robust for the uncertainty of the population distribution. In *the facility location problem*, we are given a set of weighted points and a set of candidate points on which the facilities can be located, and then, required to optimally locate the facilities on some of candidate points. Here, we suppose that the weight of a point corresponds to the number of evacuees given there, and a facility corresponds to a evacuation center. In the following, we call such a facility corresponding to a evacuation center *a sink*, and a location of sinks *a sink location* (also called *k-sink location* when the number of sinks is k).

In the problem we consider, an area in which sinks are located is modeled in a graph network, that is, every vertex is weighted and any point on an edge or a vertex is a candidate of sink. First, let us consider a simpler case in which each vertex is given a fixed weight (although we will give another setting so

that we can take the uncertainty of the population distribution into account). In order to represent the evacuation, we consider *the dynamic setting* in graph networks, which was first introduced by Ford et al. [23]. Under the dynamic setting, a vertex is weighted with *a supply* which is a set of substantial units, called *supply units*. Here, each supply unit corresponds to an evacuee given the vertex. These units can move in the network along edges. Throughout this dissertation, we denote such movement of supply units in the network by the term *flow*. Here, each edge of an input graph has a capacity which limits the rate of the flow into the edge per unit time. We call such networks under the dynamic setting *dynamic networks*. Note that in dynamic networks, we can represent the traffic congestion which may occur in the real evacuation. Then, an natural criterion of sink location is the time required to complete the evacuation. If a sink location is given in a dynamic network with the initial supplies on vertices, *evacuation time* of the sink location is defined as the minimum time required to send all supply units to the sinks. Then, *the optimal sink location problem in dynamic networks* is defined as the problem to find a sink location in a given network which minimizes the evacuation time.

However, as mentioned above, since the number of evacuees in an area may vary depending on the time, the optimal solution for a fixed assignment of vertex supplies (corresponding to a particular population distribution) may not be robust for other assignments. So, in order to take into account the uncertainty of vertex supplies, we consider *the maximum regret* for a particular sink location as another evaluation criterion assuming that for each vertex, we only know the interval of vertex supply. In this dissertation, we formulate the problem as *the minimax regret sink location problem in dynamic networks*.

Let us explain the problem with the number of sinks $k \geq 1$ in a graph network. A particular assignment of supplies to vertices is called *a scenario*, and let $\mathcal{S}$ denote a set of all possible scenarios. Given a scenario $s \in \mathcal{S}$, a k-partition of vertices of an input graph $\mathcal{P} = \{V_1, \ldots, V_k\}$ and a location of k sinks $\boldsymbol{x} = \{x_1, \ldots, x_k\}$, we assume that all supplies on V_i assigned by s are sent to x_i. Throughout this dissertation, we abuse the term k-sink location to call such a pair of a k-partition and a location of k sinks $(\boldsymbol{x}, \mathcal{P})$. Let $\Theta_i^s(\boldsymbol{x}, \mathcal{P})$ denote the minimum time required to send all supplies on V_i assigned by s to x_i, and $\Theta^s(\boldsymbol{x}, \mathcal{P})$ denote the maximum of $\Theta_i^s(\boldsymbol{x}, \mathcal{P})$ for $1 \leq i \leq k$. Letting Θ_{opt}^s denote the minimum of $\Theta^s(\boldsymbol{x}, \mathcal{P})$ for all possible $(\boldsymbol{x}, \mathcal{P})$, we define *the regret* of a k-sink location $(\boldsymbol{x}, \mathcal{P})$ under a scenario s as $R^s(\boldsymbol{x}, \mathcal{P}) = \Theta^s(\boldsymbol{x}, \mathcal{P}) - \Theta_{\mathrm{opt}}^s$.

The maximum regret is defined for a k-sink location $(\boldsymbol{x}, \mathcal{P})$ as

$$R_{\max}(\boldsymbol{x}, \mathcal{P}) = \max_{s \in \mathcal{S}} R^s(\boldsymbol{x}, \mathcal{P}).$$

Then, the goal is to find a k-sink location $(\boldsymbol{x}, \mathcal{P})$ which minimizes $R_{\max}(\boldsymbol{x}, \mathcal{P})$, called *the minimax regret sink location*. Indeed the minimax regret sink location problem is very important for the robustness of evacuation planning, we have some difficulties to solve the problems, e.g., there are many scenarios to be considered, so it may take time to solve the problems if we do exhaustively. In this dissertation, we will propose efficient algorithms for the minimax regret sink location problems in dynamic networks and show time bounds in which our algorithms can solve the problems.

1.2.1 Previous work

Dynamic networks were introduced by Ford et al. [23] and have since been extensively analyzed. A basic problem on dynamic networks is *the quickest transshipment problem*, in which given several weighted vertices with supplies (called *sources*) and several sinks on vertices in an input network, the goal is to find a flow which satisfies all of the demands as quickly as possible. For this problem, Hoppe et al. [28] considered general graph networks with several sources and several sinks, and gave a polynomial algorithm. Indeed their algorithm was the first one which achieved a polynomial running time, but it was high-order polynomial. Then, several researchers have studied the problem in restricted graph networks and developed efficient algorithms, e.g., for tree networks with a single sink, Mamada et al. [33] developed an $O(n \log^2 n)$ time algorithm, and for grid networks with a single sink where each edge has uniform capacity and uniform length, Kamiyama et al. [30] developed an $O(n \log^2 n)$ time algorithm.

In this dissertation, we are interested in the sink location problems in dynamic networks as mentioned above. The optimal sink location problem is defined as the problem to find a sink location in a given network which minimizes the evacuation time. Note that in this problem, each vertex is given a fixed supply. As far as we know, there is only one study for this problem except the results in this dissertation, that is, for the optimal 1-sink location problem in dynamic tree networks, Mamada et al. [33] developed an $O(n \log^2 n)$ time algorithm.

Recently, several researchers have studied the minimax regret facility loca-

tion problems and proposed efficient algorithms [6, 7, 10, 11, 12, 13, 32, 37, 39]. However, no one has studied the minimax regret sink location problem in dynamic networks before our studies.

1.3 Contributions of This Dissertation

The rest of this dissertation consists of six chapters. In Chapter 2, we give notations and definitions which are necessary in the subsequent discussion, and Chapter 7 concludes this dissertation. The main results of this dissertation will appear in Chapters 3-6. These chapters are divided into two parts. Part I and Part II consist of two chapters, respectively. In Part I, we consider the space exploration problems under incomplete information, and in Part II, we consider the sink location problems in dynamic networks under incomplete information. Here we give summaries of these chapters and illustrate the connection among these chapters.

1.3.1 Part I

In Part I, we will propose online algorithms for the online space exploration problems with its competitive ratios.

Chapter 3: Online Exploration Problems in Graph Networks. In Chapter 3, we consider the exploration in undirected graphs by multiple searchers. Given an undirected graph as an input, the searchers move along edges and targets to be visited are all vertices of the graph. We especially focus on cycles and trees. Also, in Chapter 3, we consider two models: *returnable* and *non-returnable* models. In the former model, a searcher can stop or change direction at any intermediate point of an edge while in the latter model, he/she can do so only at a vertex. Here, let p be the number of searchers. In Section 3.3, for cycles with $p = 2$, we propose an optimal 1.5-competitive algorithm in non-returnable model (notice that three or more searchers do not perform better than two in the case of cycles). In Section 3.4, for trees with general p in returnable model, we propose a $(p + \lfloor \log p \rfloor)/(1 + \lfloor \log p \rfloor)$-competitive algorithm under the much weaker assumption of communication ability of searchers such that searchers can only communicate when they meet or by marking the vertices they visited. Although the algorithm is essentially the same as [24], our analysis is new and much simpler, and moreover the upper bound for the competitive ratio of the algorithm [24] contains a hidden constant coefficient 4.

Thus, our result improves the constant coefficient of the upper bound of [24]. At the end of Section 3.4, we also show that the lower bound of competitive ratio for all of *greedy algorithms* is $\Omega(p/\log p)$, which improves upon the more general lower bound of $\Omega(\log p/\log\log p)$ by [20] for this restricted class. This implies that our algorithm is optimal among greedy algorithms.

Table 1.1: Table for the online exploration problems in graph networks

	1-searcher	p-searchers
Cycle	$(1+\sqrt{3})/2$ [c]	1.5 [a]
Tree	1	$(p+\lfloor\log p\rfloor)/(1+\lfloor\log p\rfloor)$ [b]
Planer graph	16 [d]	open
Graph with genus g	$16(1+2g)$ [e]	open

Contributions: [a, b] are proved in Sections 3.3 and 3.4, respectively.
Previous results: [c, d, e] have been proved by [35, 29, 34], respectively.

Chapter 4: Online Exploration Problems in Simple Polygons. In Chapter 4, we consider the exploration in polygons in the Euclidean plane by a single searcher. Given a simple polygon as an input, the searcher freely moves in the inside of the polygon and targets to be visited are all vertices of the polygon. We especially focus on simple polygons which has no hole in the inside. Also, in Chapter 4, we assume that an input polygon is explored by a single searcher. In Section 4.3, we propose an algorithm and show that it achieves a competitive ratio at most 1.219 and at least 1.040. In Section 4.4, we show that the same algorithm achieves a competitive ratio at most 1.167 and at least 1.034 restricting the class of object space to rectilinear simple polygons.

Table 1.2: Table for the online exploration problems in simple polygons

	1-searcher	p-searchers
Simple polygon	1.219 [f]	open
Rectilinear simple polygon	1.167 [g]	open
Polygon with holes	open	open

Contributions: [f, g] are proved in Sections 4.3 and 4.4, respectively.

1.3.2 Part II

In Part II, we will propose efficient algorithms for the minimax regret sink location problems in dynamic networks and show time bounds in which our algorithms can solve the problems. In the problems we consider in Part II, we assume that each edge has uniform capacity.

Chapter 5: Minimax Regret Sink Location Problems in Dynamic Path Networks. In Chapter 5, we consider the minimax regret sink location problems in dynamic path networks with uniform capacity. First, we treat the case of 1-sink location. In Section 5.3, we propose an $O(n \log n)$ time algorithm for the minimax regret 1-sink location problem. Next, we treat the case of k-sink location. In order to develop an algorithm for the minimax regret k-sink location problem, we first consider the optimal k-sink location problem. In Section 5.4, we propose an $O(kn)$ time algorithm for the optimal k-sink location problem, and in Section 5.5, we prove that the minimax regret k-sink location problem can be solved in $O(kn^3)$ time by using the algorithm proposed in Section 5.4 as a subroutine. Also, in Section 5.4, we consider the optimal k-sink location problem in a dynamic path network with general capacities and prove that the problem can be solved in $O(kn^2 \log n)$ time.

Chapter 6: Minimax Regret Sink Location Problems in Dynamic Tree Networks. In Chapter 6, we consider the minimax regret 1-sink location problem in dynamic tree networks with uniform capacity. In order to develop an algorithm for the minimax regret 1-sink location problem, we first consider the optimal 1-sink location problem. In Section 6.3, we propose an $O(n \log n)$ time algorithm for the optimal 1-sink location problem. Note that the algorithm proposed in Section 6.3 improves upon the existing time bound of $O(n \log^2 n)$ by [33] restricting the setting so that each edge has uniform capacity. In Section 6.4, we propose an $O(n^2 \log^2 n)$ time algorithm for the minimax regret 1-sink location problem, which uses the algorithm proposed in Section 6.3 as a subroutine.

Table 1.3: Table for the optimal sink location problems

	General capacities		Uniform capacity	
	1-sink	k-sink	1-sink	k-sink
Path	$O(n \log n)$ [i]	$O(kn^2 \log n)$ [i]	$O(n)$ [i]	$O(kn)$ [i]
Tree	$O(n \log^2 n)$ [m]	open	$O(n \log n)$ [k]	open
General graph	open	open	open	open

Table 1.4: Table for the minimax regret sink location problems

	General capacities		Uniform capacity	
	1-sink	k-sink	1-sink	k-sink
Path	open	open	$O(n \log n)$ [h]	$O(kn^3)$ [j]
Tree	open	open	$O(n^2 \log^2 n)$ [l]	open
General graph	open	open	open	open

Contributions: [h, i, j, k, l] are proved in Sections 5.3, 5.4, 5.5, 6.3 and 6.4, respectively.
Previous results: [m] has been proved by [33].

1.3.3 Relationship among the chapters

Figure 1.1 shows the connection among all chapters.

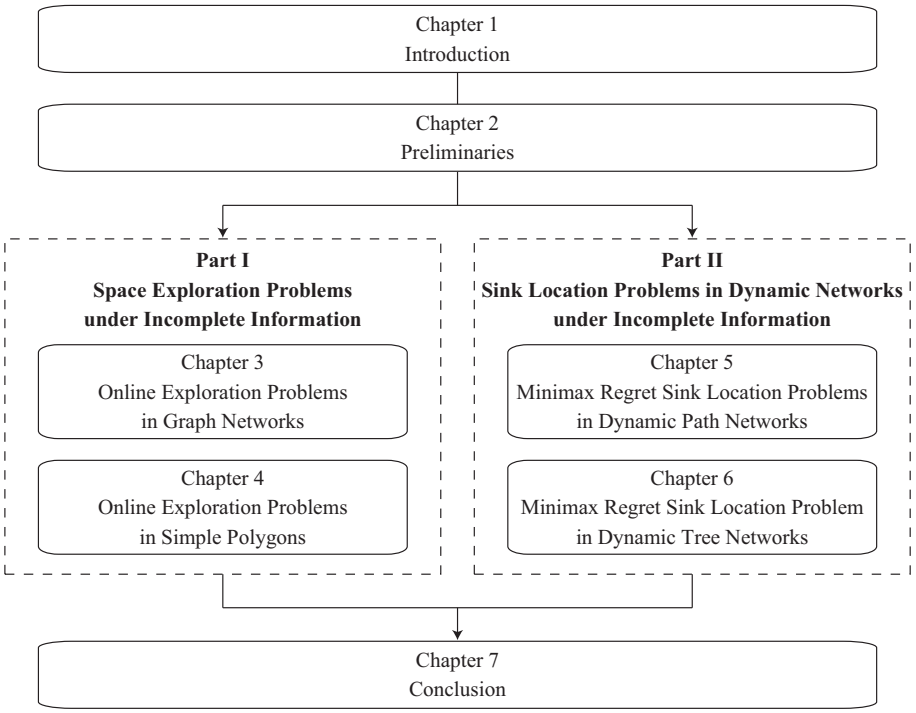

Figure 1.1: Relationship among the chapters

1.3.4 Publications

Chapter 3 is based on the paper [G]. Chapter 4 is based on the paper [F]. Chapter 5 is based on the papers [B, E]. A preliminary version of [B, E] are the papers [A, D], respectively. Chapter 6 is based on [C].

Chapter 2

Preliminaries

In this chapter, we introduce basic notations and definitions which will be used in the subsequent chapters.

2.1 Preliminaries for Part I

In this section, we give some preliminaries for Part I.

2.1.1 Competitive ratio

We again explain the concept of *competitive ratio* for online exploration algorithms. Throughout Part I, we assume that the distance traveled by each searcher per unit time is 1. Let $\mathcal{X}$ denote a class of object spaces. When an online exploration algorithm ALG is used to explore an object space $X \in \mathcal{X}$, let $\mathsf{ALG}(X)$ denote the time required to complete the exploration of X by ALG. Also, let $\mathsf{OPT}(X)$ denote the time required to complete the exploration of X by an offline optimal algorithm. Then the competitive ratio of ALG is defined as follows:

$$cr_{\mathsf{ALG}} = \sup_{X \in \mathcal{X}} \frac{\mathsf{ALG}(X)}{\mathsf{OPT}(X)}. \tag{2.1}$$

As mentioned in Chapter 1, the competitive ratio cr_{ALG} ensures how inaccurate the algorithm ALG is in the worst case comparing with an offline optimal algorithm, in other words, the worth of information for the class of object spaces in which ALG works. A lower bound of competitive ratios for all online algorithms which work in the class of object spaces ensures how hard exploring such object spaces is.

2.1.2 α-competitive algorithm

Let us consider an online algorithm ALG for an online exploration problem which requires to explore a class of object spaces $\mathcal{X}$. If the competitive ratio of ALG is at most α, we call ALG *α-competitive*. More precisely, ALG is called α-competitive if

$$\mathsf{ALG}(X) \leq \alpha \cdot \mathsf{OPT}(X) + \beta \tag{2.2}$$

holds for any $X \in \mathcal{X}$ where β is a constant.

2.1.3 Definitions for Chapter 3

In Chapter 3, we consider the exploration in an undirected graph by multiple searchers.

Undirected graph: Let $G = (V, E)$ be an undirected graph where V is a vertex set and E is an edge set. For an edge $e \in E$ where two end vertices are u and v, we use the notation uv to denote the edge e. In Chapter 3, each edge $e \in E$ is given a length, which is denoted by $l(e)$. Let L denote the sum of all edge lengths:

$$L = \sum_{e \in E} l(e). \tag{2.3}$$

A particular vertex is defined as *the origin* where the searchers are located at the beginning. Here, we abuse the notation G to denote the set of all points p on G. Then, for two points p and $q \in G$, let $d(p, q)$ denote the shortest distance between p and q in G.

Cycle: In Section 3.3, we focus on the exploration in cycles. *A cycle* is defined as an undirected graph which consists of an alternate sequence of vertices and edges $(v_1, e_1, v_2, e_2, ..., v_n, e_n)$ such that e_i connects v_i and v_{i+1} for each i with $1 \leq i \leq n$ where $v_{n+1} = v_1$. We denote an input cycle by $C = (V, E)$ in Section 3.3.

Tree: In Section 3.4, we focus on the exploration in trees. *A tree* is defined as an undirected graph in which any two vertices are connected by exactly one simple path. We denote an input tree by $T = (V, E)$ in Section 3.4. In a tree, a vertex to which only one vertex is adjacent is called *a leaf*. *A rooted tree* is defined as a tree in which a particular vertex is designated *the root*. Then, we determine the parent-children relationship between vertices in a rooted tree.

Given a rooted tree $T = (V, E)$, for a vertex $v \in V$ which is not the root, *the parent* of v is an adjacent vertex to v on the path from v to the root. Also, for a vertex $v \in V$ which is not a leaf, *a child* of v is a vertex whose parent is v, and *a descendant* of v is a vertex such that v is on the path to the root. Note that every vertex except the root has a unique parent, and a leaf does not have any descendant or child. Given a rooted tree $T = (V, E)$, for a vertex $v \in V$, let $T(v)$ denote the subtree which is induced by v and all descendants of v. For any vertices u and $v \in V$ such that v is a child of u, let $T(u; v)$ denote the subgraph induced by u and vertices of $T(v)$. In Section 3.4, $T(u; v)$ will be called *a branch* of u.

Greedy algorithm: In Section 3.4, we propose a greedy algorithm for the online exploration problem in trees. An algorithm for the online exploration problem is called *greedy* if each searcher proceeds to explore areas which have been found but have not been explored as long as there exist such areas.

Returnable and Non-returnable models: In Chapter 3, we consider two models: *returnable* and *non-returnable* models. In the former model, a searcher can stop or change direction at any intermediate point of an edge while in the latter model, he/she can do so only at a vertex.

2.1.4 Definitions for Chapter 4

In Chapter 4, we consider the exploration in a simple polygon in the Euclidean plane by a single searcher.

Definitions in the Euclidean plane: For any path A in the Euclidean plane, let $|A|$ denote the distance along A. For any two points p and q in the Euclidean plane, let pq denote the line segment between p and q. Note that from the above definitions, we denote the Euclidean distance between two points p and q by $|pq|$.

Simple polygon: *A simple polygon* is defined as a region in the Euclidean plane (including the boundary) enclosed by a closed polygonal chain with no self-intersection. *A closed polygonal chain* is defined as an alternate sequence of vertices and edges $(v_1, e_1, v_2, e_2, ..., v_n, e_n)$ such that all vertices are embedded in the Euclidean plane and each e_i is a line segment connecting v_i and v_{i+1} for each i with $1 \leq i \leq n$ where $v_{n+1} = v_1$. A closed polygonal chain is said to have

no self-intersection if only consecutive two edges (or the first and the last edges) intersect at their common endpoints. A particular point is given in the inside of an input simple polygon as *the origin* where the searcher is located at the beginning. In Chapter 4, we abuse a term *polygon* to denote a simple polygon. Given a polygon P, we call a vertex and an edge of P *a polygon vertex* and *a polygon edge*, respectively. We also call the closed polygonal chain forming P *the boundary of* P. Moreover, we abuse the notation P to denote a set of all points in the inside and on the boundary of a polygon P. Let V and E denote a set of polygon vertices and a set of polygon edges, respectively. For a polygon edge $e \in E$ where two endpoints are u and $v \in V$, we use the notation uv to denote the polygon edge e. Then, for a polygon edge $e = uv \in E$, we use the notations $|e|$ or $|uv|$ to denote the length of the edge e. Let L denote the sum of all edge lengths:

$$L = \sum_{e \in E} |e|. \tag{2.4}$$

Rectilinear simple polygon: In Section 4.4, we focus on the exploration in rectilinear simple polygons. *A rectilinear simple polygon* is defined as a simple polygon all of whose interior angles are $\pi/2$, π or $3\pi/2$ (see Figure 2.1). In Section 4.4, we abuse the term *a rectilinear polygon* to denote a rectilinear simple polygon. Polygon edges of rectilinear polygon are classified as *horizontal edges* or *vertical edges*. We assume that the searcher follows the Euclidean shortest path even if he/she is in a rectilinear polygon, that is, the searcher can freely move in the inside of a rectilinear polygon. Given a rectilinear polygon R, let us consider the minimum enclosing rectangle of R, called R'. Then, *the height* of R and *the width of* R is defined as the height of R' and the width of R', respectively.

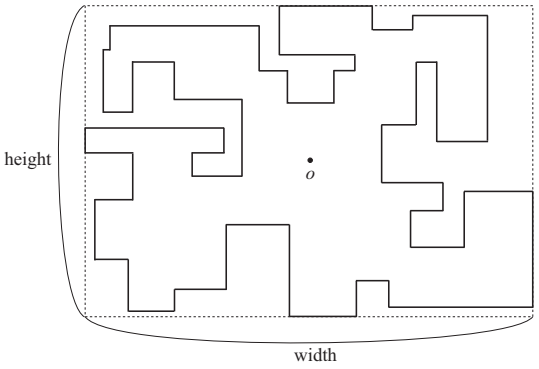

Figure 2.1: Illustration of a rectilinear polygon

2.2 Preliminaries for Part II

In this section, we give some preliminaries for Part II. In Part II, although we aim to solve sink location problems in dynamic networks with intervals of supplies on vertices, we also consider sink location problems in dynamic networks with fixed supplies on vertices as subproblems. Suppose that the capacity of an edge limits the rate of flow into the edge per unit time. Throughout Part II, we assume that if we place a sink at a vertex, all supply units of the vertex can finish the evacuation in no time. Thus, without loss of generality, we assume the number of located sinks is at most the number of vertices (otherwise, at least one sink can be located at each vertex).

2.2.1 Dynamic network with uniform capacity under fixed supplies

A dynamic network with uniform capacity under fixed supplies $\mathcal{N} = (G = (V, E), w, l, c', \tau)$ consists of an undirected graph $G = (V, E)$, a function w which associates each vertex $v \in V$ with a positive supply, a function l which associates each edge $e \in E$ with a positive length, a positive constant c' which represents the uniform capacity of edges, and a positive constant τ which represents the time required by the flow to traverse the unit distance in the network. In Part II, for a vertex $v \in V$, we abuse $w(v)$ to denote the amount of supply of v although $w(v)$ represents the supply of v as a set of substantial units.

Evacuation time of a sink location: Given a k-sink location $(\boldsymbol{x}, \mathcal{P})$ with a k-partition of vertices of an input graph $\mathcal{P} = \{V_1, \ldots, V_k\}$ and a location of k sinks $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ in a dynamic network under fixed supplies, let $\Theta_i(\boldsymbol{x}, \mathcal{P})$ denote the minimum time required to send all supplies on V_i to x_i, and $\Theta(\boldsymbol{x}, \mathcal{P})$ denote the maximum of $\Theta_i(\boldsymbol{x}, \mathcal{P})$ for $1 \leq i \leq k$:

$$\Theta(\boldsymbol{x}, \mathcal{P}) = \max\{\Theta_i(\boldsymbol{x}, \mathcal{P}) \mid 1 \leq i \leq k\}. \tag{2.5}$$

Then, we call $\Theta(\boldsymbol{x}, \mathcal{P})$ *evacuation time* of the k-sink location $(\boldsymbol{x}, \mathcal{P})$.

2.2.2 Dynamic network with uniform capacity under uncertain supplies

A dynamic network with uniform capacity under uncertain supplies $\mathcal{N} = (G = (V, E), W, l, c', \tau)$ consists of the same components as the one under fixed supplies except for a function w. Here, a function W associates each vertex $v \in V$ with an interval of supply such that $W(v) = [w^-(v), w^+(v)]$ with $0 < w^-(v) \leq w^+(v)$ instead of w.

Scenario: In a dynamic network under uncertain supplies, a particular assignment of supplies to vertices is called *a scenario*. Let $\mathcal{S}$ denote the Cartesian product of all $W(v)$ for $v \in V$, i.e., a set of scenarios:

$$\mathcal{S} = \prod_{v \in V} [w^-(v), w^+(v)]. \tag{2.6}$$

When a scenario $s \in \mathcal{S}$ is given, we use the notation $w^s(v)$ to denote the supply of each vertex $v \in V$ under the scenario s.

Evacuation time of a sink location under a scenario: Given a scenario $s \in \mathcal{S}$ and a k-sink location $(\boldsymbol{x}, \mathcal{P})$ with $\mathcal{P} = \{V_1, \ldots, V_k\}$ and $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ in a dynamic network under uncertain supplies, let $\Theta_i^s(\boldsymbol{x}, \mathcal{P})$ denote the minimum time required to send all supplies on V_i assigned by s to x_i, and $\Theta^s(\boldsymbol{x}, \mathcal{P})$ denote the maximum of $\Theta_i^s(\boldsymbol{x}, \mathcal{P})$ for $1 \leq i \leq k$:

$$\Theta^s(\boldsymbol{x}, \mathcal{P}) = \max\{\Theta_i^s(\boldsymbol{x}, \mathcal{P}) \mid 1 \leq i \leq k\}. \tag{2.7}$$

Then, we call $\Theta^s(\boldsymbol{x}, \mathcal{P})$ *evacuation time* of the k-sink location $(\boldsymbol{x}, \mathcal{P})$ under the scenario s.

Optimal evacuation time under a scenario: Given a scenario $s \in \mathcal{S}$, let Θ^s_{opt} denote the minimum of $\Theta^s(\boldsymbol{x}, \mathcal{P})$ for all possible sink locations $(\boldsymbol{x}, \mathcal{P})$:

$$\Theta^s_{\text{opt}} = \min\{\Theta^s(\boldsymbol{x}, \mathcal{P}) \mid \text{all possible } (\boldsymbol{x}, \mathcal{P})\}. \tag{2.8}$$

Regret of a sink location under a scenario: Given a scenario $s \in \mathcal{S}$ and a k-sink location $(\boldsymbol{x}, \mathcal{P})$, we define *the regret* of $(\boldsymbol{x}, \mathcal{P})$ under s as follows:

$$R^s(\boldsymbol{x}, \mathcal{P}) = \Theta^s(\boldsymbol{x}, \mathcal{P}) - \Theta^s_{\text{opt}}. \tag{2.9}$$

Maximum regret of a sink location: Given a k-sink location $(\boldsymbol{x}, \mathcal{P})$, we define *the maximum regret* of $(\boldsymbol{x}, \mathcal{P})$ as follows:

$$R_{\max}(\boldsymbol{x}, \mathcal{P}) = \max_{s \in \mathcal{S}} R^s(\boldsymbol{x}, \mathcal{P}). \tag{2.10}$$

Worst case scenario for a sink location: Given a k-sink location $(\boldsymbol{x}, \mathcal{P})$, we call a scenario $s^* \in \mathcal{S}$ *a worst case scenario* if

$$s^* = \operatorname*{argmax}_{s \in \mathcal{S}} R^s(\boldsymbol{x}, \mathcal{P}). \tag{2.11}$$

Minimax regret sink location: A sink location which minimizes $R_{\max}(\boldsymbol{x}, \mathcal{P})$ for all possible $(\boldsymbol{x}, \mathcal{P})$ is called *the minimax regret sink location.*

2.2.3 Discrete and continuous models

Dynamic networks can be considered in *discrete* and *continuous* models. In discrete model, each input value is given as an integer. Then, each supply can be regarded as a set of evacuees, and edge capacity is defined as the maximum number of evacuees who can enter an edge per unit time. On the other hand, in continuous model, each input value is given as a real number. Then, each supply can be regarded as fluid, and edge capacity is defined as the maximum amount of supply which can enter an edge per unit time. Throughout Part II, we consider the continuous model since it has more tractable properties and seems to be proper to represent the real evacuation by the great number of evacuees in large cities.

2.2.4 Definitions for Chapter 5

In Chapter 5, we consider the minimax regret sink location problems in dynamic path networks with uniform capacity. A dynamic path network consists of a path as an input graph.

Path: *A path* $P = (V, E)$ is defined as an undirected graph which consists of a vertex set V and an edge set E such that an alternate sequence of vertices and edges $(v_1, e_1, v_2, e_2, ..., e_{n-1}, v_n)$ and e_i connects v_i and v_{i+1} for each i with $1 \leq i \leq n-1$. In Chapter 5, we abuse the notation P to denote the set of all points p on P. Also, for a vertex $v_i \in P$ with $1 \leq i \leq n$, we abuse the notation v_i to denote the distance from v_1 to v_i, and for a point $p \in P$, we abuse the

notation p to denote the distance from v_1 to p. Then, we can regard P as embedded on a real line such that $v_1 = 0$. For two points $p, q \in P$ with $p < q$, let $[p, q]$ (resp. $[p, q)$, $(p, q]$ and (p, q)) denote the part of P which consists of all points $x \in P$ such that $p \leq x \leq q$ (resp. $p \leq x < q$, $p < x \leq q$ and $p < x < q$).

Evacuation time of a 1-sink location in a dynamic path network with uniform capacity under fixed supplies: Given a dynamic path network with uniform capacity under fixed supplies $\mathcal{N} = (P = (V, E), w, l, c', \tau)$, suppose that a sink is located at a point $x \in P$. Let $\Theta_L(x)$ (resp. $\Theta_R(x)$) denote the minimum time required to send all supplies on the part of P consisting of all points $p \in P$ such that $p < x$ (resp. $x < p$) to x. Letting $\Theta(x)$ denote the evacuation time of a sink location x, $\Theta(x)$ can be represented as follows:

$$\Theta(x) = \max\left\{\Theta_L(x), \Theta_R(x)\right\}. \tag{2.12}$$

For discrete model, $\Theta_L(x)$ and $\Theta_R(x)$ can be expressed by [30] as follows:

$$\Theta_L(x) = \max_i \left\{ (x - v_i)\tau + \left\lceil \frac{\sum_{1 \leq j \leq i} w(v_j)}{c'} \right\rceil - 1 \;\middle|\; v_i \in [v_1, x) \right\}, \tag{2.13}$$

$$\Theta_R(x) = \max_i \left\{ (v_i - x)\tau + \left\lceil \frac{\sum_{i \leq j \leq n} w(v_j)}{c'} \right\rceil - 1 \;\middle|\; v_i \in (x, v_n] \right\}. \tag{2.14}$$

From these, we can immediately develop the formulae for continuous model as follows:

$$\Theta_L(x) = \max_i \left\{ (x - v_i)\tau + \frac{\sum_{1 \leq j \leq i} w(v_j)}{c'} \;\middle|\; v_i \in [v_1, x) \right\}, \tag{2.15}$$

$$\Theta_R(x) = \max_i \left\{ (v_i - x)\tau + \frac{\sum_{i \leq j \leq n} w(v_j)}{c'} \;\middle|\; v_i \in (x, v_n] \right\}. \tag{2.16}$$

Here, we assume $\Theta_L(v_1) = 0$ and $\Theta_R(v_n) = 0$. In Chapter 5, for the ease of exposition, we assume that $c' = 1$ holds (the case of $c' > 1$ can be treated in essentially the same manner in continuous model). Thus, $\Theta_L(x)$ and $\Theta_R(x)$ can be redefined as follows:

$$\Theta_L(x) = \max_i \left\{ (x - v_i)\tau + \sum_{1 \leq j \leq i} w(v_j) \;\middle|\; v_i \in [v_1, x) \right\}, \tag{2.17}$$

$$\Theta_R(x) = \max_i \left\{ (v_i - x)\tau + \sum_{i \leq j \leq n} w(v_j) \;\middle|\; v_i \in (x, v_n] \right\}. \tag{2.18}$$

Evacuation time of a 1-sink location in a dynamic path network with general capacities under fixed supplies: *A dynamic network with general capacities under fixed supplies* $\mathcal{N} = (G = (V, E), w, l, c, \tau)$ consists of the same components as the one under fixed supplies except for a positive constant c'. Here, a function c associates each edge $e \in E$ with a positive capacity. In the following, we give the formulae for the evacuation time of a 1-sink location in a dynamic path network with general capacities under fixed supplies in a similar manner to (2.17) or (2.18). Given a dynamic path network with general capacities under fixed supplies $\mathcal{N} = (P = (V, E), w, l, c, \tau)$, suppose that a sink is located at a point $x \in P$. In the same manner as the case of uniform capacity, let $\Theta_L(x)$ (resp. $\Theta_R(x)$) denote the minimum time required to send all supplies on the part of P consisting of all points $p \in P$ such that $p < x$ (resp. $x < p$) to x, and let $\Theta(x)$ denote the evacuation time of a sink location x. Then, $\Theta(x)$ can be represented as (2.12). For discrete model, we prove the following formulae:

$$\Theta_L(x) = \max_i \left\{ (x - v_i)\tau + \left\lceil \frac{\sum_{1 \le j \le i} w(v_j)}{\min_{e_j \in [v_i, x]} c(e_j)} \right\rceil - 1 \;\middle|\; v_i \in [v_1, x) \right\}, \quad (2.19)$$

$$\Theta_R(x) = \max_i \left\{ (v_i - x)\tau + \left\lceil \frac{\sum_{i \le j \le n} w(v_j)}{\min_{e_j \in [x, v_i]} c(e_j)} \right\rceil - 1 \;\middle|\; v_i \in (x, v_n] \right\}. \quad (2.20)$$

In order to prove these, we only need to prove (2.20) holds ((2.19) can be symmetrically proved). Without loss of generality, we consider the following situation: we are given a dynamic path network $\mathcal{N} = (P, w, l, c, \tau)$ where P consists of an alternate sequence of vertices and edges $(v_0, e_1, v_1, e_2, ..., e_n, v_n)$, and a sink location x is given on the vertex v_0. In such a situation, we use the notation $\Theta(x, \mathcal{N})$ to denote the minimum time required to send all supplies in $\mathcal{N}$ to x, Therefore, we prove the following formula instead of (2.20):

$$\Theta(x, \mathcal{N}) = \max \left\{ (v_i - x)\tau + \left\lceil \frac{\sum_{i \le j \le n} w(v_j)}{\min_{1 \le j \le i} c(e_j)} \right\rceil - 1 \;\middle|\; 1 \le i \le n \right\}. \quad (2.21)$$

In order to prove (2.21), we show the following two lemmas.

Lemma 1. *For any i with $1 \le i \le n$,*

$$\Theta(x, \mathcal{N}) \ge (v_i - x)\tau + \left\lceil \frac{\sum_{i \le j \le n} w(v_j)}{\min_{1 \le j \le i} c(e_j)} \right\rceil - 1 \quad (2.22)$$

holds.

Lemma 2. *For some i^* with $1 \leq i^* \leq n$,*

$$\Theta(x, \mathcal{N}) \leq (v_{i^*} - x)\tau + \left\lceil \frac{\sum_{i^* \leq j \leq n} w(v_j)}{\min_{1 \leq j \leq i^*} c(e_j)} \right\rceil - 1 \tag{2.23}$$

holds.

If we prove Lemmas 1 and 2, we can immediately obtain (2.21), which implies that the formulae (2.19) and (2.20) are proved.

Proof of Lemma 1. We only prove (2.22) for a fixed i with $1 \leq i \leq n$. Let $k = \mathrm{argmin}\{c(e_j) \mid 1 \leq j \leq i\}$. We transform the original network $\mathcal{N} = (P, w, l, c, \tau)$ to $\mathcal{N}_1 = (P, w_1, l, c_1, \tau)$ such that $w_1(v_j) = 0$ for $1 \leq j \leq n$ except $j = i$ and $w_1(v_i) = \sum_{i \leq j \leq n} w(v_j)$, and $c_1(e_j) = C$ for $1 \leq j \leq n$ except $j = k$ where C is a sufficiently large constant and $c_1(e_k) = c(e_k) = \min\{c(e_j) \mid 1 \leq j \leq i\}$. We then have

$$\Theta(x, \mathcal{N}) \;\geq\; \Theta(x, \mathcal{N}_1). \tag{2.24}$$

Let us consider the evacuation to x in $\mathcal{N}_1$. All evacuees, i.e., $w_1(v_i)$ evacuees, can reach v_k at time $(v_i - v_k)\tau$, and the last evacuee leaves v_k at time $(v_i - v_k)\tau + \lceil w_1(v_i)/c_1(e_k) \rceil - 1$. Thus, we have

$$\begin{aligned} \Theta(x, \mathcal{N}_1) &= (v_i - v_k)\tau + \left\lceil \frac{w_1(v_i)}{c_1(e_k)} \right\rceil - 1 + (v_k - x)\tau \\ &= (v_i - x)\tau + \left\lceil \frac{w_1(v_i)}{c_1(e_k)} \right\rceil - 1. \end{aligned} \tag{2.25}$$

By (2.24), (2.25) and the definitions of $w_1(v_i) = \sum_{i \leq j \leq n} w(v_j)$ and $c_1(e_k) = \min\{c(e_j) \mid 1 \leq j \leq i\}$, we can obtain (2.22) for a fixed i with $1 \leq i \leq n$. We can apply the same discussion to all other i. □

Proof of Lemma 2. We transform the original network $\mathcal{N} = (P, w, l, c, \tau)$ to $\mathcal{N}_2 = (P, w, l, c_2, \tau)$ such that $c_2(e_i) = \min\{c(e_j) \mid 1 \leq j \leq i\}$ for i with $1 \leq i \leq n$. Note that $c_2(e_i) \leq c(e_i)$ holds for any i with $1 \leq i \leq n$, and $c_2(e_i) \geq c_2(e_{i+1})$ also holds for any i with $1 \leq i \leq n-1$. We then have

$$\Theta(x, \mathcal{N}) \;\leq\; \Theta(x, \mathcal{N}_2). \tag{2.26}$$

Let us consider the evacuation to x in $\mathcal{N}_2$. We define an *evacuee of* i as a evacuee who is initially located at a vertex v_i for i with $1 \leq i \leq n$, the *first evacuee of* i as an evacuee of i who leaves v_i at time 0 and reaches x first in

all evacuees of i, and the *last evacuee of* i as an evacuee of i who leaves v_i at time $\lceil w(v_i)/c_2(e_i) \rceil - 1$ and reaches x last in all evacuees of i. Without loss of generality, we assume that an evacuee who first reaches a vertex first leaves the vertex. When an evacuee reaches some vertex v at time t, he/she is said to *stop at* v if he/she has to leave v at time $t+1$ or later than $t+1$.

Now, suppose that the last evacuee of n moves to v_i with some i with $1 \leq i \leq n-1$ without any stopping and stops at v_i. Let t_1 denote the time when the last evacuee of n reaches v_i, that is,

$$t_1 = (v_n - v_i)\tau + \left\lceil \frac{w(v_n)}{c_2(e_n)} \right\rceil - 1. \tag{2.27}$$

We show that from time 0 to $t_1 - 1$, $c_2(e_i)$ evacuees leave v_i at each time. Suppose otherwise. Let t' denote the first time such that $0 \leq t' \leq t_1 - 1$ when the number of evacuees who leave v_i is less than $c_2(e_i)$. Since $c_2(e_i) \geq c_2(e_j)$ holds for any j with $i+1 \leq j \leq n$, the number of evacuees who reach v_i at time $t'+1$ or later than $t'+1$ is always at most $c_2(e_i)$, that is, each evacuees who reach v_i at time $t'+1$ or later than $t'+1$ must not stop at v_i, which contradicts the assumption that the last evacuee of n stops at v_i. Therefore, at time t_1, there remain $\sum_{i \leq j \leq n} w(v_j) - c_2(e_i)t_1$ evacuees at v_i. Let t_2 denote the time when the last evacuee of n leaves v_i, that is,

$$t_2 = t_1 + \left\lceil \frac{\sum_{i \leq j \leq n} w(v_j) - c_2(e_i)t_1}{c_2(e_i)} \right\rceil - 1 = \left\lceil \frac{\sum_{i \leq j \leq n} w(v_j)}{c_2(e_i)} \right\rceil - 1. \tag{2.28}$$

From the above discussion, we can derive the following claim.

Claim 1. *Suppose that the last evacuee of n moves to v_i with some i with $1 \leq i \leq n-1$ without any stopping and stops at v_i. Let $\mathcal{N}_3 = (P, w_3, l, c_2, \tau)$ be a network transformed from $\mathcal{N}_2$ such that $w_3(v_j) = w(v_j)$ for $1 \leq j \leq i-1$, $w_3(v_i) = \sum_{i \leq j \leq n} w(v_j)$ and $w_3(v_j) = 0$ for $i+1 \leq j \leq n$. Then,*

$$\Theta(x, \mathcal{N}_2) = \Theta(x, \mathcal{N}_3) \tag{2.29}$$

holds.

When considering the evacuation to x in $\mathcal{N}_3$, if the last evacuee of i stops at some vertex after leaving v_i before reaching x, we can apply the same transformation as Claim 1 without changing the minimum completion time. After we repeatedly apply the same transformation, we can obtain a network $\mathcal{N}^* = (P, w^*, l, c_2, \tau)$ such that for some i^* with $1 \leq i^* \leq n$, $w^*(v_j) = w(v_j)$ for

$1 \leq j \leq i^* - 1$, $w_3(v_{i^*}) = \sum_{i^* \leq j \leq n} w(v_j)$ and $w_3(v_j) = 0$ for $i^* + 1 \leq j \leq n$, and the last evacuee of i^* does not stop at any vertex after leaving v_{i^*} before reaching x. Thus, we have

$$\Theta(x, \mathcal{N}_2) = \Theta(x, \mathcal{N}^*) = (v_{i^*} - x)\tau + \left\lceil \frac{\sum_{i^* \leq j \leq n} w(v_j)}{c_2(e_{i^*})} \right\rceil - 1. \tag{2.30}$$

By (2.26), (2.30) and the definition of $c_2(e_{i^*}) = \min\{c(e_j) \mid 1 \leq j \leq i^*\}$, we can obtain (2.23). □

From (2.19) and (2.20), we can immediately develop the formulae for continuous model as follows:

$$\Theta_L(x) = \max_i \left\{ (x - v_i)\tau + \frac{\sum_{1 \leq j \leq i} w(v_j)}{\min_{e_j \in [v_i, x]} c(e_j)} \;\middle|\; v_i \in [v_1, x) \right\}, \tag{2.31}$$

$$\Theta_R(x) = \max_i \left\{ (v_i - x)\tau + \frac{\sum_{i \leq j \leq n} w(v_j)}{\min_{e_j \in [x, v_i]} c(e_j)} \;\middle|\; v_i \in (x, v_n] \right\}. \tag{2.32}$$

2.2.5 Definitions for Chapter 6

In Chapter 6, we consider the minimax regret 1-sink location problems in dynamic tree networks with uniform capacity. A dynamic tree network consists of a path as an input graph.

Tree: *A tree* $T = (V, E)$ is defined as an undirected graph which consists of a vertex set V and an edge set E such that any two vertices are connected by exactly one simple path in T and $|V| = n$. For an edge $e \in E$ where two end vertices are u and v, we use the notation uv to denote the edge e. In Chapter 6, we abuse the notation T to denote the set of all points p on T. For two points p and $q \in T$, let $d(p, q)$ denote the distance between p and q in T. For a vertex $v \in V$, let $\delta(v)$ denote a set of vertices adjacent to v, and for a point $p \in T$ which is not at a vertex but on an edge $uv \in E$, let $\delta(p)$ denote a set of two vertices u and v. Let us consider a point $p \in T$. If p is not at a vertex but on an edge $uv \in E$, let us regard p as a new vertex of T and split uv into two new edges up and pv. Then, let $T(p)$ be a rooted tree made from T such that each edge has a natural orientation towards the root p, and for any vertex $v \in V$, let $T(p, v)$ be a subtree of $T(p)$ rooted at v.

Evacuation time of a 1-sink location in a dynamic tree network with uniform capacity under fixed supplies: Given a dynamic tree network

with uniform capacity under fixed supplies $\mathcal{N} = (T = (V, E), w, l, c', \tau)$, suppose that a sink is located at a point $x \in T$. For a vertex $u \in \delta(x)$, let $\Theta(x, u)$ denote the minimum time required to send all supplies on $T(x, u)$ to x. Letting $\Theta(x)$ denote the evacuation time of a sink location x, $\Theta(x)$ can be represented as follows:

$$\Theta(x) = \max\{\Theta(x, u) \mid u \in \delta(x)\}. \tag{2.33}$$

Here, we only need to consider $\Theta(x, \hat{u})$ for $\hat{u} = \text{argmax}\{\Theta(x, u) \mid u \in \delta(x)\}$. Suppose that there are n' vertices in $T(x, \hat{u})$ named $v_1(= \hat{u}), v_2, \ldots, v_{n'}$ such that $d(x, v_j) \leq d(x, v_{j+1})$ for $1 \leq j \leq n' - 1$. For discrete model, Kamiyama et al. [30] have observed that the value of $\Theta(x, \hat{u})$ does not change if x and all v_j for $1 \leq j \leq n'$ are relocated on a line with the same capacity so that $d(x, v_j)$ for $1 \leq j \leq n'$ remain the same (see Figure 2.2), and $\Theta(x, \hat{u})$ can be represented as follows:

$$\Theta(x, \hat{u}) = \max_{1 \leq i \leq n'} \left\{ d(x, v_i)\tau + \left\lceil \frac{\sum_{i \leq j \leq n'} w(v_j)}{c'} \right\rceil - 1 \right\}. \tag{2.34}$$

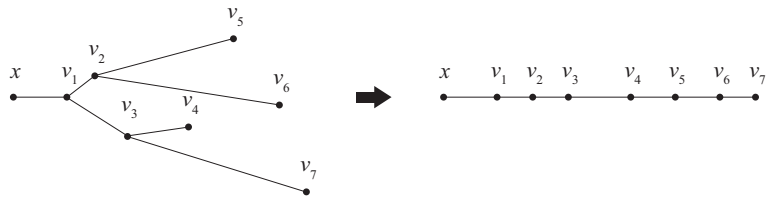

Figure 2.2: Vertices of the tree can be relocated on a line with the same capacity

For the completeness, we now see why this formula holds. We first define a *group* as a set of evacuees who simultaneously reach x from $\hat{u}$ and the *size* of a group as the number of evacuees in the group. Suppose that a group whose size is less than c' reaches x at time t'. Then, we call a group which first reaches x after t' a *leading group* (see Fig. 2.3). We also call a group which first reaches

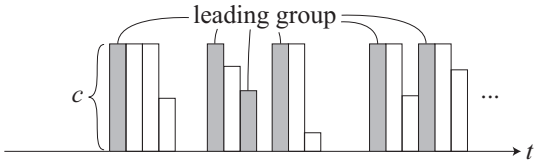

Figure 2.3: The size of groups reaching x from $\hat{u}$ for each time

x after time 0 a leading group. Let t_{last} denote the time when the last group reaches x (i.e., the whole evacuation finishes at t_{last}). Suppose that a leading group reaches x at time t'' and there is no leading group which reaches x after t'' until t_{last}. Then, we call a leading group reaching x at t'' the *last leading group* and a set of groups reaching x from t'' to t_{last} the *last cluster*. In order to derive $\Theta(x, \hat{u})$, we only need to observe the last cluster. We notice that all evacuees of a leading group are located at vertices whose distance from x are the same at time 0, and they all reach x without being blocked. Suppose that all evacuees of the last leading group are located at vertices $v_l, v_{l+1}, \ldots, v_{l+k}$ such that $d(x, v_l) = d(x, v_{l+1}) = \ldots = d(x, v_{l+k})$ at time 0. Then, the last leading group reaches x at time $d(x, v_l)\tau$, and then, all groups except ones which belong to the last cluster have already reached x. If $d(x, v_l)\tau < t_{\text{last}}$, the size of a group reaching x at each time $t \in [d(x, v_l)\tau, t_{\text{last}} - 1]$ is exactly c' because of definition of the last leading group. Therefore, $\Theta(x, \hat{u})$ can be represented as follows:

$$\Theta(x, \hat{u}) = d(x, v_l)\tau + \left\lceil \frac{\sum_{l \le i \le n'} w(v_i)}{c'} \right\rceil - 1.$$

Note that this still holds for the case of $d(x, v_l)\tau = t_{\text{last}}$. We next see that the right hand of the formula (2.34) is the lower bound for $\Theta(x, \hat{u})$. For all evacuees located at $v_j, \ldots, v_{n'}$ with some integer $1 \le j \le n'$, the time of $d(x, v_j)\tau + \lceil \sum_{j \le i \le n'} w(v_i)/c \rceil - 1$ is at least required to complete the evacuation to x, thus we have

$$\Theta(x, \hat{u}) \ge d(x, v_j)\tau + \left\lceil \frac{\sum_{j \le i \le n'} w(v_i)}{c'} \right\rceil - 1$$

for any integer $1 \le j \le n'$. From the above discussion, we can derive the formula (2.34).

For continuous model, we can observe the same property as above, that is, $\Theta(x, \hat{u})$ can be represented as follows:

$$\Theta(x, \hat{u}) = \max_{1 \le i \le n'} \left\{ d(x, v_i)\tau + \frac{\sum_{i \le j \le n'} w(v_j)}{c'} \right\}. \tag{2.35}$$

In Chapter 6, for the ease of exposition, we assume that $c' = 1$ holds (the case of $c' > 1$ can be treated in essentially the same manner in continuous model).

Thus, $\Theta(x, \hat{u})$ can be redefined as follows:

$$\Theta(x, \hat{u}) = \max_{1 \le i \le n'} \left\{ d(x, v_i)\tau + \sum_{i \le j \le n'} w(v_j) \right\}. \tag{2.36}$$

Evacuation time of a 1-sink location in a dynamic tree network with general capacities under fixed supplies: In contrast to a dynamic tree network with uniform capacity or a dynamic path network with general capacities, any formula has never been known for the evacuation time of a 1-sink location in a dynamic tree network with general capacities.

Part I

Space Exploration Problems under Incomplete Information

Chapter 3

Online Exploration Problems in Graph Networks

3.1 Introduction

In this chapter, we consider the online exploration problems in undirected graphs by multiple searchers. Restriction of network structures to undirected graphs is useful since such structure often appears in modeling building corridors and city streets. In this chapter, we will develop two efficient algorithms for the online exploration problems in cycles and trees, respectively, and analyze their competitive ratios.

3.2 Outline

In Section 3.3, we consider the online exploration problem in cycles by two searchers with non-returnable model, and propose an online algorithm with an upper bound analysis and a lower bound analysis. In Section 3.4, we consider the online exploration problem in trees by general p searchers with returnable model, and propose an online algorithm with an upper bound analysis and a lower bound analysis.

3.3 Online Exploration Problems in Cycles

In this section, we consider the online exploration problem in cycles by two searchers with non-returnable model. We are given a cycle $C = (V, E)$ and the origin $o \in V$ where two searchers are initially located. Then, the goal of

the two searchers is to visit all vertices in V so that each vertex is visited by at least one searcher and to come back to the origin o. For this problem, we propose an online algorithm, which is called ALE (Avoiding-Longest-Edge).

We first notice that any exploration algorithm must visit the vertex in V farthest to the origin o. Letting m denote the midpoint in C such that $d(o, m) = L/2$ where $L = \sum_{e \in E} l(e)$ (refer (2.3)), there are the following two cases: (i) m does not coincide with a vertex, and (ii) m coincides with a vertex.

In the former case (i), let e_m be the edge which contains m. Note that one of two endpoints of e_m is the farthest to o. Therefore in this case, an optimal offline algorithm sends the two searchers clockwise and counterclockwise from o to the two endpoints of e_m and back to o. Similarly, we can consider an online algorithm in returnable model. It sends the two searchers clockwise and counterclockwise and makes them return as soon as both of them enter the same edge e_m, and has a competitive ratio of 1. In the former case (ii), m is the farthest vertex, and trivially both of an optimal offline algorithm and optimal online algorithms in returnable model send the two searchers clockwise and counterclockwise from o to m. From these cases, for returnable model, we can obtain an optimal online algorithm which has a competitive ratio of 1.

We below explain an online algorithm ALE. The algorithm works in two phases. In the first phase, the searchers visit all vertices, and in the second phase they return to the origin. The overall idea of the first phase is to avoid traversing the longest edge. Namely, it will construct a minimum spanning tree of C. Thus, the algorithm can be viewed as Prim's algorithm for minimum spanning trees. Let s_1 and s_2 denote two searchers, who are initially at the origin o (see Figure 3.1(a)). The searchers s_1 and s_2 travel along the cycle in clockwise and counterclockwise direction, respectively. Let e_1 be the next edge that s_1 would traverse in its clockwise walk, and let e_2 be the next edge that s_2 would traverse in its counterclockwise walk. If $l(e_1) \leq l(e_2)$, s_1 traverses e_1 while s_2 remains at the same place, otherwise s_2 traverses e_2 while s_1 remains at the same place.

We repeat the same strategy until both searchers see the same edge which must have the maximum length $l_{\max}$ where $l_{\max} = \max_{e \in E} l(e)$ (see Figure 3.1(c)). We call uniquely this longest edge $e_{\max}$. At this stage, all vertices are visited, and the first phase ends. At this point, the searchers have knowledge of the entire graph. Then, they return to the origin o via shortest paths.

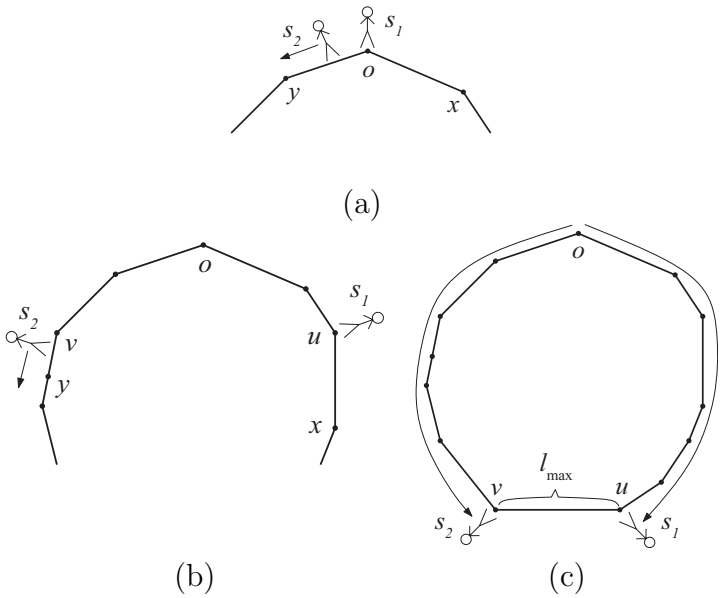

Figure 3.1: First phase: (a) initial stage; (b) intermediate stage; (c) final stage

3.3.1 Upper bound analysis

We show an upper bound of the competitive ratio of algorithm ALE. We prove the following theorem.

Theorem 1. *Algorithm* ALE *is* 1.5*-competitive.*

Proof. Let d_i for $i = 1, 2$ denote the distance traveled by s_i in the first phase. Recall that at the end of the first phase, only the edge $e_{\max}$ is unexplored. If $d_i > L/2$ holds for $i = 1$ or 2, then in the second phase he/she should return to the origin by traversing the entire cycle (see Figure3.2(b)). Otherwise he/she backtracks to the origin along the path taken in the first phase (see Figure 3.2(a)). We now show an upper bound of the competitive ratio of ALE. There are the following two cases: [Case 1] $e_{\max}$ contains m (Figure 3.2(a)), and [Case 2] $e_{\max}$ does not contain m (Figure 3.2(b)).

[Case 1]: In this case, an optimal offline algorithm sends the two searchers clockwise and counterclockwise from o to the two endpoints of e_m and back to o. Thus, $\mathsf{OPT}(C) = \max\{2d_1, 2d_2\}$ holds. On the other hand, $\mathsf{ALE}(C) = d_1 + d_2 + \max\{d_1, d_2\}$ holds. Therefore, the competitive ratio for Case 1 is

$$\frac{\mathsf{ALE}(C)}{\mathsf{OPT}(C)} = \frac{d_1 + d_2 + \max\{d_1, d_2\}}{\max\{2d_1, 2d_2\}} \leq \frac{3\max\{d_1, d_2\}}{2\max\{d_1, d_2\}} = \frac{3}{2}.$$

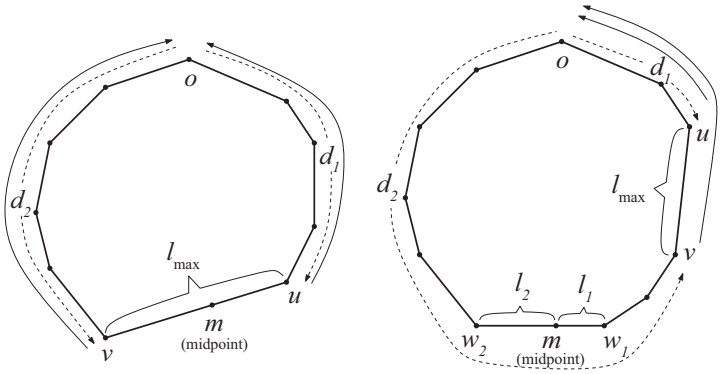

Figure 3.2: Second phase: (a) Case 1; (b) Case 2

[Case 2]: Let $e = w_1w_2$ denote the edge that contains m (in clockwise orientation). Also let $l_1 = d(w_1, m)$ and $l_2 = d(w_2, m)$. Assume without loss of generality that $e_{\max}$ is on the right part of the cycle (see Figure 3.2(b)). Since the time required by s_2 to return to o from v is $l_{\max} + d_1$ and it dominates the time required by s_1 to return to o, $\mathsf{ALE}(C) = 2d_1 + d_2 + l_{\max} = L + d_1$ holds. Also, as discussed at the beginning of Section 3.3, we have $\mathsf{OPT}(C) = \max\{L - 2l_1, L - 2l_2\} = L - 2\min\{l_1, l_2\}$. Note that $d_1 + l_{\max} + l_1 \leq L/2$, and so $d_1 \leq L/2 - l_{\max} - l_1$ hold. Because $l_{\max} \geq l_1 + l_2 \geq 2\min\{l_1, l_2\}$ and $l_1 \geq \min\{l_1, l_2\}$, we have $d_1 \leq L/2 - 3\min\{l_1, l_2\}$. Thus, $\mathsf{ALE}(C) \leq 3L/2 - 3\min\{l_1, l_2\}$ and

$$\frac{\mathsf{ALE}(C)}{\mathsf{OPT}(C)} \leq \frac{3L/2 - 3\min\{l_1, l_2\}}{L - 2\min\{l_1, l_2\}} = \frac{3}{2}.$$

By the above discussion, we obtain this theorem. □

3.3.2 Lower bound analysis

We prove that for the online exploration problem in cycles, any online algorithm never have a competitive ratio less than $1.5 - \delta$, which implies that ALE is an optimal online algorithm for the problem. We prove the following theorem.

Theorem 2. *For the online exploration problem in cycles by two searchers with non-returnable model, any online algorithm never have a competitive ratio less than* $1.5 - \delta$ *with any* $\delta > 0$.

Proof. Let N be a sufficiently large integer, and let $\varepsilon = 1/N$. We show that the competitive ratio of an arbitrary algorithm ALG is at least $1.5(1 - \varepsilon)$. Ini-

tially, an adversary provides two edges ou counterclockwise, and ov_1 clockwise from o, whose lengths are 1 and ε, respectively. We assume that s_1 traverses $ov_1, v_1v_2, v_2v_3, \ldots$ in this order, and the length of v_iv_{i+1} is set to ε where $v_0 = o$. Note that sometimes s_1 may go back to o and may traverse ou. Then, two cases can happen: [Case 1] ou is not traversed before time $(N-3)\varepsilon = 1-3\varepsilon$ (see Figure3.3(a)), and [Case 2] ou is traversed at time $d < 1 - 3\varepsilon$ (see Figure3.3(b)).

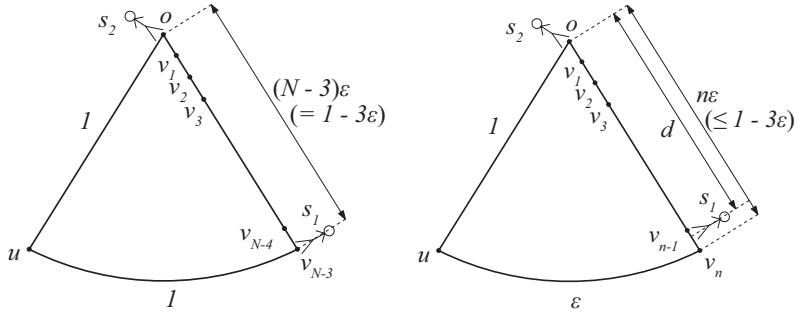

Figure 3.3: Input cycles given by an adversary: (a) Case 1; (b) Case 2

[Case 1]: In this case, an adversary connects v_{N-3} and u by an edge with length of 1 at time $1-3\varepsilon$. Then, since u can be visited at or after time $2-3\varepsilon$, $\mathsf{ALG}(C) \geq 3-3\varepsilon$ holds while $\mathsf{OPT}(C) = 2$ holds. Thus, we have

$$\frac{\mathsf{ALG}(C)}{\mathsf{OPT}(C)} \geq \frac{3}{2}(1-\varepsilon).$$

[Case 2]: Suppose that $(n-1)\varepsilon \leq d < n\varepsilon$ with some n $(1 \leq n \leq N-3)$. In this case, an adversary connects v_n and u by an edge with length of ε at time d. Note that s_1 or s_2 leaves o to u at time d in this case. Then, this searcher has to visit u at time $d+1$, and can return to o at time $d+1+(n+1)\varepsilon$ if following the counterclockwise path from u to o, which is shorter than the length of ou. Thus, $\mathsf{ALG}(C) \geq d+1+(n+1)\varepsilon$ holds while $\mathsf{OPT}(C) = 2(n+1)\varepsilon$ holds. From $n \leq N-3$, that is, $1 \geq (n+3)\varepsilon$ and $d \geq (n-1)\varepsilon$, we derive

$$\frac{\mathsf{ALG}(C)}{\mathsf{OPT}(C)} \geq \frac{d+1+(n+1)\varepsilon}{2(n+1)\varepsilon} \geq \frac{(n-1)\varepsilon+(n+3)\varepsilon+(n+1)\varepsilon}{2(n+1)\varepsilon} = \frac{3}{2}.$$

By the above discussion, we obtain this theorem. □

3.4 Online Exploration Problems in Trees

In this section, we consider the online exploration problem in trees by general p searchers with returnable model. We are given a tree $T = (V, E)$ and the origin $o \in V$ where p searchers are initially located. Then, the goal of the searchers is to visit all vertices in V so that each vertex is visited by at least one searcher and to come back to the origin o. For this problem, we propose an online algorithm, which is called BEER (Branch Evenly, Explore, Return).

We define the state of an exploration of a branch as follows. A branch is called *finished* if all vertices of the branch have been visited by at least one searcher, and there is no searcher in the branch except possibly its root. It is called *under exploration* if there is at least one searcher in the branch except its root, and it is called *unexplored* if no vertices of the branch except its root has been visited by any searcher, and there is no searcher in the branch except its root.

Algorithm BEER is a slight modification of the one by [24]. Our algorithm is *greedy*, that is, when a searcher is at a vertex v and if there are some non-finished branches of v, he/she will proceed to visit a non-finished branch of v without stopping. Also, if v is a leaf or if all branches of v are finished, he/she immediately returns to the parent of v. Note that, in the exploration by any greedy algorithm, every branch takes exactly one of the three previously described states at any time (however, for a non-greedy algorithm, a branch may not take any of these three states; for instance, there may be a branch whose vertices are partially visited but no searcher exists therein).

Before describing the algorithm, we shall make the following assumptions which help to clarify the understanding and to make the analysis easier.

Assumption: (I) The input tree T is binary, (II) searchers do not meet at any vertex.

In the case of a non-binary tree, in order to meet the assumption (I), we can transform it into a binary tree by introducing edges whose weights are zero into the original tree. Namely, our algorithm performs the exploration by regarding a non-binary tree as a binary tree transformed by the above manner. For the assumption (II), we will explain why this does not lose generality after the proof of Lemma 4. Under these assumptions, BEER performs the exploration of the tree by greedy strategy and the following two rules.

Rule 1: Suppose that a group G of searchers arrives at a vertex u which has two children v_1 and v_2 such that none of $T(u; v_1)$ and $T(u; v_2)$ are

finished. Then G will be divided into two subgroups G_1 and G_2 such that at the next time instant, the number of searchers in $T(u; v_1)$ is as equal as possible to that in $T(u; v_2)$, and G_1 and G_2 start to explore branches $T(u; v_1)$ and $T(u; v_2)$, respectively.

Rule 2: Suppose that a group G of searchers is moving forward in T and it encounters another group G' which is coming from the opposite direction. Then, G goes back towards the root together with G' by reversing the direction.

Note that in Rule 2 and in the followings, the word "forward" means "in the direction from the root to the leaves".

Compared with cycles, we may assume a much weaker communication model such that searchers can only communicate when they meet and by reading and writing messages at vertices they visited. When a searcher newly enters into an unfinished branch $T(u; v)$, he/she writes the message at u that $T(u; v)$ is under exploration as well as the number of searchers exploring the branch. When a searcher goes back from a child v, he/she writes a message at u to inform that the branch $T(u; v)$ is finished. (As will be shown in Lemma 3, a searcher does not leave $T(u; v)$ unless it is finished.) By reading the message written at a vertex, a searcher can decide where to go next.

3.4.1 Upper bound analysis

The goal of this subsection is to prove the following theorem.

Theorem 3. *Algorithm* BEER *is* $(p + \lfloor \log p \rfloor)/(1 + \lfloor \log p \rfloor)$*-competitive.*

The proof assumes p is a power of two. However, this assumption will be removed at the end of the proof. We first need a lemma, which applies to any greedy algorithm.

Lemma 3. *Let t' be the time when all branches of v become finished. Then, no searcher leaves $T(v)$ before time t'.*

Proof. Suppose otherwise. Among the p' searchers who explore $T(v)$ in total, let s be the one who exits $T(v)$ for the first time. For $p' = 1$, this lemma is trivially proved from the definition of "greedy". For $p' \geq 2$, when s leaves $T(v)$, at least one searcher is remaining at some place of $T(v)$ except v. Then there is a non-finished branch of v and thus any greedy algorithm must force s to explore $T(v)$. This is a contradiction. □

We now analyze the competitive ratio of BEER. Let $t_0 = 0$ and $t_1, t_2 \ldots$ be all distinct times (in increasing order) at which some group reaches some vertex of T or two groups meet, during the execution of BEER. Let v_{last} be the leaf that the algorithm visits last, or one of them if it is not unique. Let t_{last} be the time at which v_{last} is visited (we will also write "last" for the integer index of the time t_{last}). We call the path from o to v_{last} the *backbone*, and call a branch whose root is on the backbone and is edge-disjoint from the backbone a *rib*. Let $r_1, r_2, \ldots,$ denote the roots of ribs arranged in increasing order of the distance from the root. Notice that since we assume that an input tree is binary, $r_1, r_2, \ldots,$ are all distinct. Ribs with roots $r_1, r_2, \ldots,$ are denoted by $S_1, S_2, \ldots,$ respectively (see Figure 3.4).

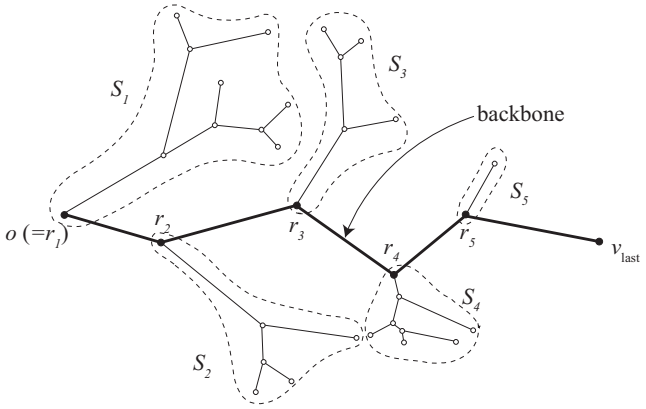

Figure 3.4: Backbone and ribs

Let a be a positive integer. Then for $1 \leq a \leq$ last, let I_a denote the time interval $[t_{a-1}, t_a]$. When a searcher s is on the backbone both at time t_{a-1} and t_a, we say that s is *moving on the backbone* at I_a. When a searcher s is in a rib S_j both at time t_{a-1} and t_a, we say that s *belongs to* S_j at I_a. Each searcher is either moving on the backbone or belonging to some rib at each interval. For each interval I_a, we define a *cluster* as a set of searchers (i) who belong to the same rib at I_a or (ii) who are moving together on the backbone during I_a. Note that a cluster reconfiguration may occur at each t_a for $0 \leq a \leq \text{last} - 1$. In the following discussion, for a searcher s, we abuse the notation s to denote a point in G where he/she is located. When s belongs to a rib S_j at I_a, the level of s (at I_a) is defined to be $d(r, r_j)$. When s is moving on the backbone at I_a, the level of s is defined to be $d(r, s)$ (by this definition, the level is increasing during I_a since no searcher goes up the backbone towards o before t_{last}). Note that the level of each member of a cluster is the same. The level of a cluster is defined to be that of a member of the cluster. Suppose that there exist $k(\geq 2)$

clusters at I_a, then let $C_1, C_2, ..., C_k$ denote clusters at I_a such that for any $1 \leq i \leq j \leq k$, the level of C_i is at most that of C_j at any time during I_a.

Lemma 4. *Suppose that at time interval I_a with $1 \leq a \leq$ last, there exist $k(\geq 2)$ clusters. Then the following equation holds.*

$$|C_i| = \frac{p}{2^i} \quad \textit{for} \quad 1 \leq i \leq k-1 \quad \textit{and} \quad |C_k| = |C_{k-1}|. \tag{3.1}$$

Proof. First of all, let us consider the case where the cluster C_i is in rib S_h at I_a, and is on the backbone at I_{a+1} for $1 \leq a \leq \text{last} - 1$. This means that C_i arrives at the root r_h of S_h at time t_a after finishing the exploration of S_h. (Note that all searchers of C_i arrive at r_h at the same time by Lemma 3, Rule 2 and the greedy property.) From the assumption (II), C_i does not meet any other cluster at r_h at time t_a. So, in this case the cluster reconfiguration does not occur for C_i.

Therefore, the cluster configuration changes only if a cluster moving on the backbone arrives at the root of a rib and then by Rule 1, the cluster is split into two clusters. Recall that p is a power of 2. Let $C_i(I_a)$ denote the cluster C_i at time interval I_a. The proof is done by the induction on a. The base case is trivial from Rule 1 and assumption (I).

Assuming that for an arbitrary a such that $1 \leq a \leq \text{last} - 1$, (3.1) holds at $[0, t_a]$, we will prove (3.1) holds at I_{a+1}. There are three cases where the cluster configuration changes. Suppose that at time t_a, the cluster C_i moving on the backbone arrives at a vertex u which has two children (say, v_1 and v_2). Assume without loss of generality that $T(u; v_1)$ is a rib while $T(u; v_2)$ contains the backbone. Note that in the following discussion, we omit the trivial case that one of the branches is already finished since in this case, the cluster is not split and then the cluster configuration does not change.

[Case 1]: Both of $T(u; v_1)$ and $T(u; v_2)$ are unexplored at I_a. Then $i = k$ must hold. When $|C_k(I_a)| = 1$, (3.1) clearly holds at I_{a+1}. Otherwise, from Rule 1, $C_k(I_a)$ is split into two clusters of equal size, i.e., $|C_k(I_{a+1})| = |C_{k+1}(I_{a+1})| = p/2^k$, respectively. Then by induction hypothesis, (3.1) clearly holds at I_{a+1}.

[Case 2]: Exactly one of $T(u; v_1)$ and $T(u; v_2)$ is unexplored and the other is under exploration at I_a. In case $T(u; v_1)$ is unexplored, there must be only one searcher in $T(u; v_2)$ (except u) at I_a because otherwise more than one searcher visited u before t_a, and thus by Rule 1 they must have been split

into two clusters such that one of two should have visited $T(u; v_1)$, which is a contradiction. Then the cluster of one searcher that is exploring $T(u; v_2)$ at I_a must have the highest level, and hence $i = k-1$ holds. Therefore, $|C_i(I_a)| = 1$ holds from the induction hypothesis. In this case, $C_i(I_a)$ enters into $T(u; v_1)$ from Rule 1, then for $i = k-1, k$, $C_i(I_{a+1}) = C_i(I_a)$ holds and thus (3.1) clearly holds at I_{a+1}. In case $T(u; v_2)$ is unexplored, we can apply the same argument (although $C_k(I_{a+1}) = C_{k-1}(I_a)$ and $C_{k-1}(I_{a+1}) = C_k(I_a)$).

[Case 3]: Both of $T(u; v_1)$ and $T(u; v_2)$ are under exploration at I_a. Then, $C_i(I_a)$ is split into two halves such that one half enters into a rib S_h and is merged with the cluster $C_{i+1}(I_a)$ which is exploring S_h, and the other portion goes along the backbone and forms a new cluster. From the definition, searchers which belong to S_h at I_a and those which enter into S_h at t_a form a cluster $C_i(I_{a+1})$ and the other portion moving along the backbone forms a new cluster $C_{i+1}(I_{a+1})$. Then from the induction hypothesis,

$$\begin{aligned} |C_i(I_a)| &= p/2^i \\ |C_{i+1}(I_a)| &= p/2^{i+1}. \end{aligned} \tag{3.2}$$

Thus from the discussion given above and (3.2), we have

$$\begin{aligned} |C_i(I_{a+1})| &= |C_i(I_a)|/2 + |C_{i+1}(I_a)| = p/2^i \\ |C_{i+1}(I_{a+1})| &= |C_i(I_a)|/2 = p/2^{i+1}. \end{aligned} \tag{3.3}$$

This proves that (3.1) holds at I_{a+1}.

Note that in Case 1, the number of clusters increases by 1 (except the case that $C_k(I_a)$ consists of only one searcher) and in the other cases, the number of clusters remains unchanged even if these cases simultaneously occur for other clusters at the same time. □

In the proof of Lemma 4, if we drop the assumption (II), some two clusters, i.e., one cluster moving on the backbone and the other which belonged to a rib at the previous interval, may meet at the root of the rib and then be merged into a new cluster because the levels of two clusters become the same. In this case, Lemma 4 does not hold any more. However, by introducing tie-breaking rule such that we regard these two clusters as different ones, Lemma 4 can still hold. Therefore the assumption (II) does not lose generality.

Furthermore, from Lemma 4, we have the following corollary.

Corollary 1. *Suppose there exist k clusters at I_a with $1 \leq a \leq \text{last}$. Then*

$$k + \log |C_k| = 1 + \log p. \tag{3.4}$$

From these lemmas and the corollary, we now prove Theorem 3. For the sake of the analysis, we will place a number of weighted tokens on each edge as follows: two tokens of weight $l(e)$ are placed on each *rib edge* e, and $1 + \log p$ tokens of weight $l(e)$ are placed on each *backbone edge* e. Note that we mean "backbone edges" and "rib edges" as those on the backbone and the ribs, respectively. We consider as these tokens are collected by searchers during the exploration by the following rules. Note that, since searchers actually cannot distinguish whether an edge belongs to a rib or the backbone, these rules are not parts of the execution of BEER but introduced just for the analysis.

Collecting rules:
(C1) For a rib edge e, whenever a number of searchers simultaneously enter e in the forward direction for the first time, exactly one searcher among them collects one token from e.
(C2) For a rib edge e, whenever a number of searchers simultaneously enter e in the backward direction for the first time, exactly one searcher among them collects one token from e.
(C3) For a backbone edge e, whenever the cluster of the highest level C_k enters e in the forward direction, each of $1 + \log |C_k|$ searchers in this cluster collects one token from e (this is possible because $|C_k| \geq 1 + \log |C_k|$).
(C4) For a backbone edge e, whenever a cluster except that of the highest level enters e in the forward direction, exactly one searcher in this cluster collects one token from e.

Because the weight of a token is always equal to the length of the edge it is placed on, we can regard a searcher who grabs a token of weight $l(e)$ when entering an edge e as if he/she receives one token of unit weight per unit time on average for the entire time while traversing e. From the way the tokens are placed and the collecting rules, we have the following observation. (i) While a cluster belongs to a rib, (we can consider that) there always exists at least one searcher in the cluster who is receiving one token of unit weight per unit time on average by the following reason. If all leaves in the rib have not been visited, there exists an edge such that at least one searcher is traversing the edge in the forward direction for the first time. Otherwise, i.e., after visiting all leaves while the cluster belongs to the rib, there exists an edge such that at

least one searcher is traversing the edge in the backward direction for the first time. Note that by the greedy property, each edge is traversed exactly once in a backward direction and all searchers in the cluster simultaneously arrive at the root of the rib; (ii) For a cluster moving on the backbone, there are enough tokens on each backbone edge to collect by the following reason. Suppose that at time t', the cluster of the highest level moving on the backbone arrives at an endpoint u of a backbone edge $e = (u, v)$ where v is a child of u (this cluster would be divided into two clusters if u has two children), and that at $t' + \varepsilon$ for an infinitesimal $\varepsilon(> 0)$, there are k clusters in total. Then at $t' + \varepsilon$, C_k has already entered e and collects $1 + \log |C_k|$ tokens from e by (C3). Moreover at $t' + \varepsilon$, the level of C_k is $d(r, u) + \varepsilon$ and there are $k - 1$ clusters whose levels are less than $d(r, u) + \varepsilon$. Since the number of clusters increases only if the cluster of the highest level is split (from the proof of Lemma 4), the number of clusters whose levels are less than $d(r, u) + \varepsilon$ never increases after $t' + \varepsilon$ and decreases by 1 whenever some cluster enters e after $t' + \varepsilon$. Thus, even if the cluster configuration changes, e is entered by at most $k-1$ clusters after $t' + \varepsilon$, then by (C3) and (C4), at most $k + \log |C_k|$ tokens are collected in total from e. From Corollary 1, we have $k + \log |C_k| = 1 + \log p$. Therefore there are enough tokens on each backbone edge.

From the above observation, Collecting rules and Corollary 1, we can consider that as long as $0 \leq t \leq t_{\text{last}}$, at least $1 + \log p$ searchers are always receiving one token of unit weight per unit time on average, thus at least $(1 + \log p)t_{\text{last}}$ tokens are collected in total. Then, since the total weight of all tokens is $2(L - d) + (1 + \log p)d$ where $L = \sum_{e \in E} l(e)$ (refer (2.3)) and d is the length of the backbone, we have the following.

$$(1 + \log p)t_{\text{last}} \leq 2(L - d) + (1 + \log p)d = 2L + (\log p - 1)d. \tag{3.5}$$

Let $d_{\max} = \max\{d(r, v) \mid v \in V\}$, then the cost of BEER obviously satisfies $\mathsf{BEER}(T) \leq t_{\text{last}} + d_{\max}$. Thus from (3.5) and $d \leq d_{\max}$,

$$\mathsf{BEER}(T) \leq \frac{2L + (\log p - 1)d}{1 + \log p} + d_{\max} \leq \frac{2(L + d_{\max} \log p)}{1 + \log p}. \tag{3.6}$$

Now remember we assumed p is a power of two. We can remove this assumption by only using p' searchers where p' is the largest power of two such that $p' \leq p$, i.e., $p - p'$ searchers remain together at o during the entire exploration. Then

$\log p' = \lfloor \log p \rfloor$, thus from (3.6), the cost of the algorithm would be

$$\mathsf{BEER}(T) \leq \frac{2(L + d_{\max} \log p')}{1 + \log p'} = \frac{2(L + d_{\max} \lfloor \log p \rfloor)}{1 + \lfloor \log p \rfloor} \tag{3.7}$$

It is known (see [36] or easily observed) that the cost of an optimal offline algorithm satisfies $\mathsf{OPT}(T) \geq \max\{2L/p, 2d_{\max}\}$. Combining this with (3.7), we obtain

$$\begin{aligned} \frac{\mathsf{BEER}(T)}{\mathsf{OPT}(T)} &\leq \min\left\{ \frac{2(L + d_{\max} \lfloor \log p \rfloor)}{\frac{2L}{p}(1 + \lfloor \log p \rfloor)}, \frac{2(L + d_{\max} \lfloor \log p \rfloor)}{2d_{\max}(1 + \lfloor \log p \rfloor)} \right\} \\ &= \min\left\{ \frac{p}{L} \cdot \frac{L + d_{\max} \lfloor \log p \rfloor}{1 + \lfloor \log p \rfloor}, \frac{1}{d_{\max}} \cdot \frac{L + d_{\max} \lfloor \log p \rfloor}{1 + \lfloor \log p \rfloor} \right\} \\ &\leq \frac{p + \lfloor \log p \rfloor}{1 + \lfloor \log p \rfloor}, \end{aligned} \tag{3.8}$$

which implies that Theorem 3 has been proved.

3.4.2 Lower bound analysis

In the previous works, Fraigniaud et al. [24] gave a lower bound of $2 - 1/p$ for the competitive ratio of any deterministic algorithm for trees with general p. Moreover Dynia et al. [20] improved it to $\Omega(\log p / \log \log p)$. In this section, we show a better lower bound for any greedy algorithm. Note that for $p = 2$, a lower bound of $2 - 1/p$ by [24] is equal to our upper bound in Section 3.4.1, thus it is sufficient to consider the case of $p \geq 3$.

Theorem 4. *For the online exploration problem in trees by p searchers with returnable model, any greedy algorithm never have a competitive ratio less than $\lceil p/(1 + \lfloor \log p \rfloor) \rceil - \delta$ with any $\delta > 0$.*

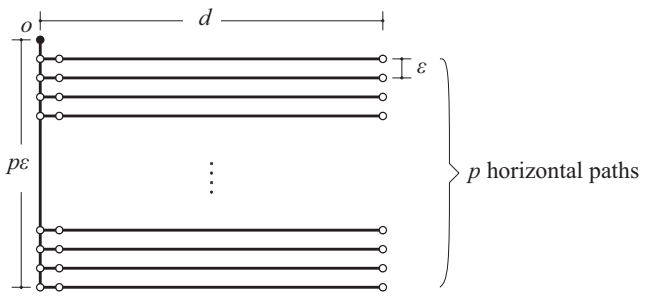

Figure 3.5: Input tree given by an adversary

Proof. In this proof, we consider an arbitrary greedy algorithm called GRD. An adversary gives the input tree illustrated in Figure 3.5. A vertex of degree three is called a *T-junction*. The input tree consists of a single vertical path of length $p\varepsilon$ and p horizontal paths of length d. The vertical path consists of p edges of length ε. A horizontal path consists of two edges of lengths ε and $d-\varepsilon$, respectively. All three edges incident to every T-junction have length ε. First of all, all p searchers start from o and move down to the first T-junction. At this point, p searchers are divided into two groups. Since the two branches are indistinguishable, the adversary can force at least half of p searchers to choose the horizontal path. Likewise, every time searchers arrive at a T-junction, the adversary forces at least half of them to choose the horizontal path. Finally, at some T-junction, it happens that all remaining searchers choose the horizontal path. When these searchers return to the T-junction after exploring the horizontal path, all the other searchers arrive at this T-junction at the same time by the construction of the input tree. Then, the exploration starts from the T-junction again in the same manner. Suppose that all searchers meet at the T-junction $n-1$ times ($n \geq 1$). Then for an integer k ($1 \leq k \leq n-1$), let t_k be the time instant when all searchers meet at the T-junction for the k-th time. Also let $t_0 = 0$ and t_n be the time when all searchers return to o. For $1 \leq k \leq n$ the exploration executed from t_{k-1} to t_k is called the *stage* k. Note that there is at least one horizontal path which is explored at the stage n since all searchers are at a T-junction at t_{n-1} (there is the unexplored part of the input tree below the T-junction).

Since at least one horizontal path is searched during each stage, $t_k - t_{k-1} \geq 2d$ holds for k ($1 \leq k \leq n$). Therefore, we have $\mathsf{GRD}(T) = t_n = \sum_{k=1}^{n}(t_k - t_{k-1}) \geq 2nd$. Since in each stage at most $1+\lfloor \log p \rfloor$ horizontal paths are explored, n must be an integer satisfying $n(1+\lfloor \log p \rfloor) \geq p$. Also, the cost of an offline optimal exploration algorithm clearly satisfies $\mathsf{OPT}(T) \leq 2d+2p\varepsilon$. We thus obtain

$$\frac{\mathsf{GRD}(T)}{\mathsf{OPT}(T)} \geq \frac{2nd}{2d+2p\varepsilon} = n\left(1-\frac{p\varepsilon}{d+p\varepsilon}\right) \geq \left\lceil \frac{p}{1+\lfloor \log p \rfloor} \right\rceil \left(1-\frac{p\varepsilon}{d+p\varepsilon}\right).$$

By letting ε be sufficiently small, we prove the theorem. □

3.5 Conclusion

In Section 3.3, we considered the online exploration problem in cycles by two searchers with non-returnable model. We proposed an online algorithm and proved that the algorithm is 1.5-competitive. Also, we proved that this ratio is tight by showing a matching lower bound. In Section 3.4, we considered the online exploration problem in trees by general p searchers with returnable model. Note that we considered the much weaker assumption of communication ability of searchers, that is, searchers can only communicate when they meet or by marking the vertices they visited. We proposed an online algorithm and proved that the algorithm is $(p + \lfloor \log p \rfloor)/(1 + \lfloor \log p \rfloor)$-competitive. Also, we proved that a lower bound of the competitive ratio for all greedy algorithms is $\Omega(p/\log p)$, which implies that our algorithm is optimal among greedy algorithms.

Chapter 4

Online Exploration Problems in Simple Polygons

4.1 Introduction

In this chapter, we consider the exploration in simple polygons in the Euclidean plane by a single searcher. Restriction of object space to the inside of simple polygon is useful since such structure often appears in modeling the inside of buildings, underground shopping areas and so on. In this chapter, we will develop an efficient algorithms for the online exploration problems in simple polygons and rectilinear simple polygons, respectively, and analyze an upper bound and a lower bound of the competitive ratio for each class.

4.2 Outline

In Section 4.3, we consider the online exploration problem in simple polygons by a single searcher, and propose an online algorithm with an upper bound analysis and a lower bound analysis. In Section 4.4, we consider the online exploration problem in rectilinear simple polygons by a single searcher, and give an upper bound analysis and a lower bound analysis for the same algorithm as in Section 4.3 in the case of restricting the class of object space to rectilinear simple polygons.

4.3 Online Exploration Problems in Simple Polygons

In this section, we consider the online exploration problem in simple polygons by a single searcher. We are given a simple polygon P consisting of a set of polygon vertices V and a set of polygon edges E, and the origin $o \in P$ where a single searcher is initially located. Then, the goal of the searcher is to visit all vertices in V and come back to the origin o. For this problem, we propose an online algorithm, which is called AOE (Avoiding-One-Edge).

4.3.1 Properties of simple polygons

For a polygon edge $e \in E$, let v_e^1 and v_e^2 denote the endpoints of e such that v_e^1 precedes v_e^2 in clockwise order. For any two points p and $q \in P$, let $sp(p, q)$ denote the shortest path from p to q which lies in the inside of P. Note that $sp(p, q) = sp(q, p)$ and $|pq| \leq |sp(p, q)|$ hold. Furthermore, for any two polygon vertices u and v, let $bp(u, v)$ denote the clockwise path along the boundary of P from u to v. For a point $x \in P$ and a polygon edge $e \in E$, let $T(x, e)$ denote the tour composed of paths $sp(x, v_e^2)$, $bp(v_e^2, v_e^1)$ and $sp(v_e^1, x)$, and $|T(x, e)|$ be its length. Then, from $|T(x, e)| = |sp(x, v_e^1)| + |sp(x, v_e^2)| + |bp(v_e^2, v_e^1)|$ and $|bp(v_e^2, v_e^1)| = L - |e|$, $|T(x, e)| = L + |sp(x, v_e^1)| + |sp(x, v_e^2)| - |e|$ holds where $L = \sum_{e \in E} |e|$ (refer (2.4)). The term $|sp(x, v_e^1)| + |sp(x, v_e^2)| - |e|$ represents the increase of the length from L, and thus we define

$$inc(x, e) = |sp(x, v_e^1)| + |sp(x, v_e^2)| - |e|. \tag{4.1}$$

Note that $|T(x, e)| = L + inc(x, e)$. Let $e_{opt} \in E$ be a polygon edge satisfying the following equation.

$$inc(o, e_{opt}) = \min_{e \in E} inc(o, e). \tag{4.2}$$

In the offline version of this problem, we will prove below that $T(o, e_{opt})$ is the optimal tour.

Lemma 5. *For the offline exploration problem in a polygon P, the tour length of an offline optimal algorithm satisfies the following.*

$$\mathsf{OPT}(P) = L + inc(o, e_{opt}).$$

Proof. Let a permutation $\pi : \{1, ..., n\} \to \{1, ..., n\}$ denote the sequence of visiting polygon vertices for the searcher. Namely the searcher visits polygon vertices in the order of $v_{\pi(1)}, v_{\pi(2)}, ..., v_{\pi(n)}$ (see Figure 4.1). Let T_π denote the tour composed of paths $sp(o, v_{\pi(1)}) \to sp(v_{\pi(1)}, v_{\pi(2)}) \to ... \to sp(v_{\pi(n)}, o)$, and $|T_\pi|$ be its length. Note that each polygon vertex must be visited in accordance with the order given by only π even if it may happen that T_π passes v_i earlier than specified by π when v_i is contained in $sp(v_{\pi(h)}, v_{\pi(h+1)})$ for some $h < \pi^{-1}(i)$. In this case, even if T_π passes through v_i in $sp(v_{\pi(h)}, v_{\pi(h+1)})$, we consider v_i is not visited by this part of T_π. If we show $|T_\pi| \geq |T(o, e_{opt})|$ for any π, then $\mathsf{OPT}(P) \geq |T(o, e_{opt})|$ is shown. Furthermore, since $\mathsf{OPT}(P) \leq |T(o, e_{opt})|$ clearly holds, the lemma is proved.

At first we define an undirected graph $G = (V', E')$ from T_π as follows. Let V' be composed of polygon vertices, self-intersection points of T_π and the origin o. Also let E' be composed of line segments between consecutive vertices in V' along T_π. However we must make parallel edges where the searcher traverses an edge more than once. Note that G is Eulerian. Let E'_1 denote a set of outermost edges of G (see Figure 4.2) and $G_1 = (V(E'_1), E'_1)$. Then $V(E'_1)$ contains V and G_1 is clearly Eulerian. There are two cases depending on a position of o.

[Case 1]: $o \in V(E'_1)$. We regard G_1 as the clockwise tour from o, and without loss of generality we can assume that for some adjacent polygon vertices, say $v_i, v_{i+1} \in V$ with $1 \leq i \leq n$, o is on the path from v_i to v_{i+1} on G_1. Note that the length of the shortest path from o to v_{i+1} on G_1 is at least $|sp(o, v_{i+1})|$, the length of the clockwise path from v_{i+1} to v_i on G_1 is at least $|bp(v_{i+1}, v_i)|$

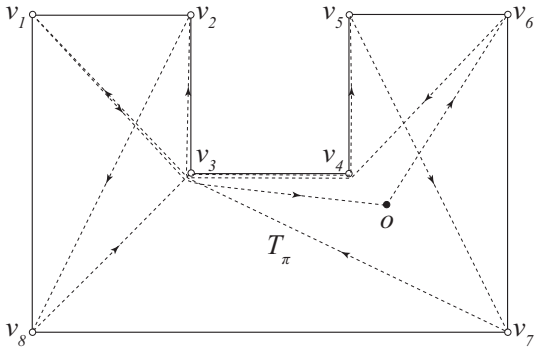

Figure 4.1: Example of π = [6 2 8 3 4 5 7 1]

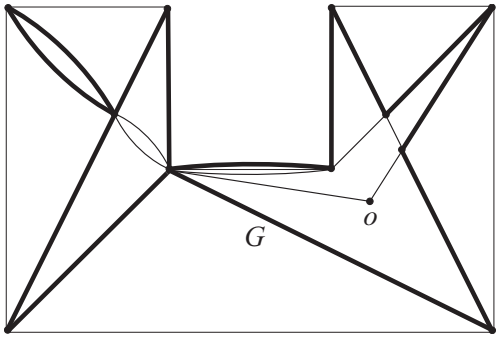

Figure 4.2: Illustration of outermost edges of G (represented by thick lines)

(since this path on G_1 visits clockwise all polygon vertices and $bp(v_{i+1}, v_i)$ is the shortest path from v_{i+1} to v_i which visits clockwise all polygon vertices), and the length of the shortest path from v_i to o on G_1 is at least $|sp(v_i, o)|$. Thus,

$$\begin{aligned} |T_\pi| &\geq |G_1| \geq |sp(o, v_{i+1})| + |bp(v_{i+1}, v_i)| + |sp(v_i, o)| \\ &= |T(o, e_i)| \geq |T(o, e_{opt})|. \end{aligned}$$

[Case 2]: $o \notin V(E'_1)$. Let $E'_2 = E \setminus E'_1$ and $G_2 = (V(E'_2), E'_2)$, and then let G_3 denote the connected component of G_2 which contains o. Let $u \in V'$ be an intersection point of G_1 and G_3, and we assume that u is on the path from v_i to v_{i+1} in G_1 for some $v_i, v_{i+1} \in V$ with $1 \leq i \leq n$. Clearly G_3 is also Eulerian, hence there are paths on G_3, p_1 from o to u and p_2 from u to o, which share no edge. In the same way as Case 1, we obtain $|G_1| \geq |sp(u, v_{i+1})| + |bp(v_{i+1}, v_i)| + |sp(v_i, u)|$. Thus,

$$\begin{aligned} |T_\pi| &\geq |p_1| + |G_1| + |p_2| \\ &\geq |sp(o, u)| + |sp(u, v_{i+1})| + |bp(v_{i+1}, v_i)| + |sp(v_i, u)| + |sp(u, o)| \\ &\geq |T(o, e_i)| \geq |T(o, e_{opt})|. \end{aligned}$$

□

Given a polygon P, for any two points x and $y \in P$, we say that y is *visible* from x if the line segment xy contains no points of the outside of P. Then, for a point $x \in P$, *the visibility polygon* $VP(x)$ is defined as follows:

$$VP(x) = \{y \in P \mid y \text{ is visible from } x\}. \tag{4.3}$$

Note that an edge of a visibility polygon is not necessarily a polygon edge (see Figure 4.3). For a polygon vertex b and a point $x \in P$, we call b *a blocking*

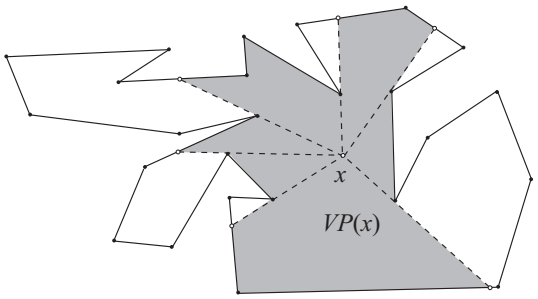

Figure 4.3: Illustration of a visibility polygon $VP(x)$ (represented by the shaded area)

vertex with respect to x if b is visible from x and there is an unique polygon edge incident to b such that any point on the edge except b is not visible from x. Let c be a point where the extension of the line segment xb towards b first intersects the boundary of P. Then, we call c *a virtual vertex* and the line

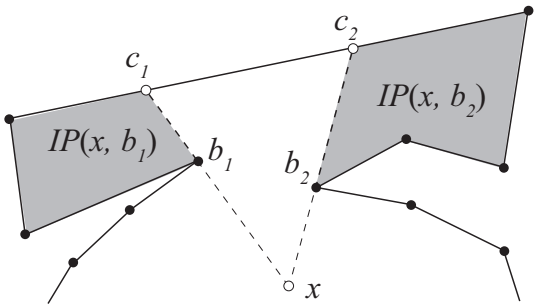

Figure 4.4: Illustration of blocking vertices b_1 and b_2, virtual vertices c_1 and c_2, cut edges b_1c_1 and b_2c_2, a virtual edge c_1c_2, and invisible polygons $IP(x, b_1)$ and $IP(x, b_2)$

segment bc *a cut edge*. Note that a virtual vertex may not coincide with any polygon vertex although a blocking vertex is always a polygon vertex. Also, let $\hat{e}$ be a polygon edge containing some virtual vertices then we regard a visible part of $\hat{e}$ as a new edge, which we call *a virtual edge*. Here, a cut edge bc divides P in two areas, a polygon which contains $VP(x)$ and the other not. We call the latter area *the invisible polygon* $IP(x, b)$ (see Figure 4.4).

Notice that $VP(x)$ and $IP(x, b)$ share a cut edge bc. We assume that there is a blocking vertex b with respect to the origin o since otherwise an optimal solution can be found by Lemma 5. Then we have the following lemma.

Lemma 6. *For an invisible polygon $IP(o, b)$ defined by a blocking vertex b, let $e \in E$ be a polygon edge such that both of its endpoints are in $IP(o, b)$, and $w \in V$ be a polygon vertex adjacent to b which is not in $IP(o, b)$. Then*

$$inc(o, bw) < inc(o, e).$$

Proof. First, we remark a simple fact. Let x, y, z be points in P such that both line segments xz and zy are lying in the inside of P. Then the following inequality obviously holds.

$$|sp(x, y)| \leq |xz| + |zy|. \tag{4.4}$$

Notice that the equality holds only when either (i) $sp(x, y)$ is a line segment xy and z is on xy, or (ii) $sp(x, y)$ is composed of two line segments xz and zy, i.e., y is not visible from x and z is a blocking vertex with respect to x.

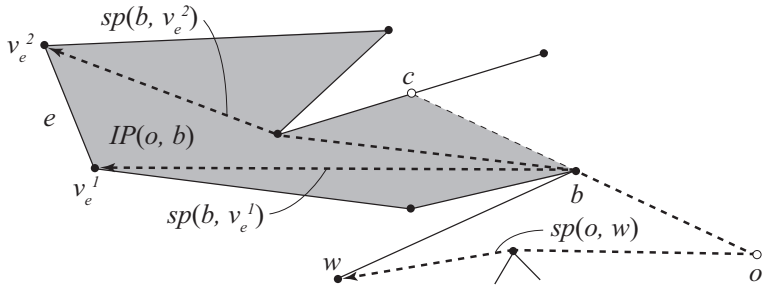

Figure 4.5: Illustration of $sp(b, v_e^1)$, $sp(b, v_e^2)$ and $sp(o, w)$ (the shaded area represents $IP(o, b)$)

See Figure 4.5. From the above observation and since b is visible from o, i.e., $|sp(o, b)| = |ob|$,

$$|sp(o, w)| < |ob| + |bw| = |sp(o, b)| + |bw|. \tag{4.5}$$

Besides, from the triangle inequality with respect to b, v_e^1 and v_e^2,

$$inc(b, e) = |sp(b, v_e^1)| + |sp(b, v_e^2)| - |e| \geq 0. \tag{4.6}$$

Furthermore both $sp(o, v_e^1)$ and $sp(o, v_e^2)$ pass through b. Hence, we have

$$\begin{aligned} |sp(o,b)| + |sp(b,v_e^1)| &= |sp(o,v_e^1)|, \text{ and} \\ |sp(o,b)| + |sp(b,v_e^2)| &= |sp(o,v_e^2)|. \end{aligned} \tag{4.7}$$

Thus,

$$\begin{aligned} inc(o,bw) &= |sp(o,b)| + |sp(o,w)| - |bw| \\ &< |sp(o,b)| + |sp(o,b)| + |bw| - |bw| && \text{(from (4.5))} \\ &\leq 2|sp(o,b)| + |sp(b,v_e^1)| + |sp(b,v_e^2)| - |e| && \text{(from (4.6))} \\ &= inc(o,e) && \text{(from (4.7))} \end{aligned}$$

holds. □

For e_{opt} defined by (4.2), the following corollary is immediate from Lemma 6.

Corollary 2. *For an invisible polygon $IP(o,b)$ defined by a blocking vertex b, let $e \in E$ be a polygon edge both endpoints of which are in $IP(o,b)$. Then e cannot be e_{opt}.*

Based on Corollary 2, candidates of e_{opt} are polygon edges or virtual edges in $VP(o)$.

4.3.2 Algorithm

In what follows, we propose an online algorithm AOE. By Lemma 5, an offline optimal algorithm chooses a polygon edge e_{opt} which satisfies (4.2). But we cannot obtain the whole information about P. So, the seemingly best strategy based on the information of $VP(o)$ is to choose an edge of $VP(o)$ in the same way as an offline optimal algorithm, assuming that there is no invisible polygon, namely $P = VP(o)$. Let E_1^* denote a polygon edge set composed of all $e \in E$ such that both endpoints of e are visible from o, E_2^* denote a set of virtual edges on the boundary of $VP(o)$ and $E^* = E_1^* \cup E_2^*$. Also for a virtual edge $e \in E_2^*$, endpoints of e are labeled as v_e^1, v_e^2 in clockwise order around o and as in (4.1), let $inc(o,e)$ denote the value of $|sp(o,v_e^1)| + |sp(o,v_e^2)| - |e|$. Let $e^* \in E^*$ be an edge satisfying the following equation.

$$inc(o,e^*) = \min_{e \in E^*} inc(o,e) \tag{4.8}$$

Then Algorithm AOE is described as follows.

Step 1: Choose $e^* \in E^*$ satisfying (4.8).

Step 2: If $e^* \in E_1^*$ then let $\hat{e} = e^*$, else let $\hat{e}$ be a polygon edge containing e^*.

Step 3: Follow the tour $T(o, \hat{e})$.

4.3.3 Upper bound analysis

In order to prove an upper bound of the competitive ratio of AOE, we first show the following lemma.

Lemma 7. *Let x be a point on the boundary of P and e^* be an edge satisfying (4.8). If x is visible from the origin o, then*

$$\frac{inc(o, e^*)}{2} \leq |ox|.$$

Proof. Let $e' \in E^*$ be an edge of $VP(o)$ containing x. Then from (4.4), we have $|ox| \geq |sp(o, v_{e'}^1)| - |xv_{e'}^1|$ and $|ox| \geq |sp(o, v_{e'}^2)| - |xv_{e'}^2|$. Therefore, we obtain

$$\begin{aligned} 2|ox| &\geq |sp(o, v_{e'}^1)| + |sp(o, v_{e'}^2)| - |xv_{e'}^1| - |xv_{e'}^2| \\ &= |sp(o, v_{e'}^1)| + |sp(o, v_{e'}^2)| - |e'| \geq inc(o, e^*), \end{aligned}$$

namely $|ox| \geq inc(o, e^*)/2$. □

Furthermore, we show a lemma which plays a crucial role in our analysis.

Lemma 8. *Let L be the length of the boundary of P and e^* be an edge satisfying (4.8). Then the following inequality holds.*

$$L \geq \pi \cdot inc(o, e^*). \tag{4.9}$$

Proof. Let C be a circle centered at the origin o with the radius of $inc(o, e^*)/2$. From Lemma 7, any polygon edge does not intersect C. Thus L is greater than the length of the circumference of C, namely

$$L \geq 2\pi \cdot \frac{inc(o, e^*)}{2} = \pi \cdot inc(o, e^*)$$

holds. □

Theorem 5. *The competitive ratio of Algorithm* AOE *is at most* 1.319.

Proof. The tour length of Algorithm AOE obviously satisfies

$$\mathsf{AOE}(P) = L + inc(o, e^*).$$

On the other hand, the tour length of an offline optimal algorithm satisfies $\mathsf{OPT}(P) = L + inc(o, e_{opt})$ holds from Lemma 5. By the triangle inequality, $inc(o, e_{opt}) \geq 0$, namely $\mathsf{OPT}(P) \geq L$ holds. Thus we have

$$\frac{\mathsf{AOE}(P)}{\mathsf{OPT}(P)} \leq \frac{L + inc(o, e^*)}{L} = 1 + \frac{inc(o, e^*)}{L}.$$

From this and (4.9),

$$\frac{\mathsf{AOE}(P)}{\mathsf{OPT}(P)} \leq 1 + \frac{inc(o, e^*)}{\pi \cdot inc(o, e^*)} = 1 + \frac{1}{\pi} \leq 1.319$$

is obtained. □

Theorem 5 gives an upper bound of the competitive ratio. In the followings, we will obtain a better bound by a detailed analysis. First, we improve a lower bound of $\mathsf{OPT}(P)$. Note that for some points $x, y, z \in P$ such that both y and z are visible from x and the line segment yz is lying in the inside of P, we call $\angle yxz$ the *visual angle* at x formed by yz.

Lemma 9. *For an edge $e^* \in E^*$ satisfying* (4.8)*, let $d = inc(o, e^*)$ and θ ($0 \leq \theta \leq \pi$) be a visual angle at o formed by a visible part of e_{opt}. Then*

$$\mathsf{OPT}(P) \geq L + d - d \sin \frac{\theta}{2}. \tag{4.10}$$

Proof. We first show the following claim.

Claim 2. *Let $b_1 \in V$* (resp. b_2) *be the polygon vertex visible from o such that the path $sp(o, v^1_{e_{opt}})$* (resp. $sp(o, v^2_{e_{opt}})$) *passes through b_1* (resp. b_2) (see Figure 4.6)*. Then*

$$inc(o, e_{opt}) \geq |ob_1| + |ob_2| - |b_1 b_2|. \tag{4.11}$$

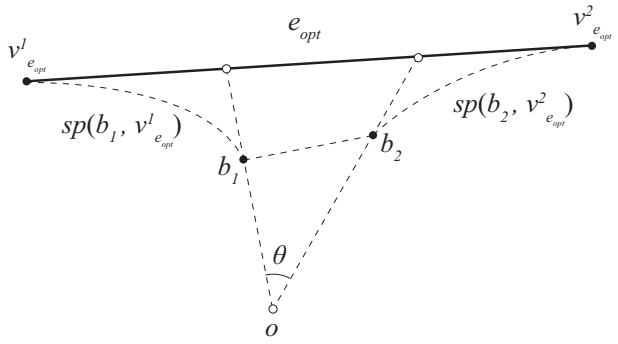

Figure 4.6: Illustration of a visible part of e_{opt} from o

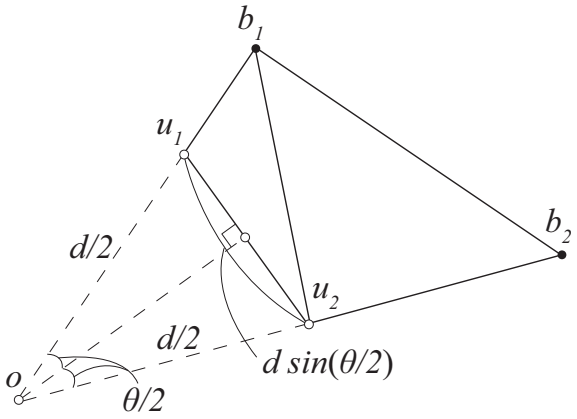

Figure 4.7: Illustration of u_1 and u_2

Proof. This claim is obtained from $|sp(o, v^1_{e_{opt}})| = |ob_1| + |sp(b_1, v^1_{e_{opt}})|$, $|sp(o, v^2_{e_{opt}})| = |ob_2| + |sp(b_2, v^2_{e_{opt}})|$ and $|e_{opt}| = |sp(v^1_{e_{opt}}, v^2_{e_{opt}})| \leq |sp(b_1, v^1_{e_{opt}})| + |b_1b_2| + |sp(b_2, v^2_{e_{opt}})|$. □

From (4.11), we have

$$\mathsf{OPT}(P) = L + inc(o, e_{opt}) \geq L + |ob_1| + |ob_2| - |b_1b_2|. \tag{4.12}$$

Furthermore b_1 and b_2 satisfy $|ob_1| \geq d/2$ and $|ob_2| \geq d/2$ from Lemma 7. Hence there exist points u_1, u_2 on line segments ob_1, ob_2 such that $|ou_1| = |ou_2| = d/2$ (see Figure 4.7). Then, from the triangle inequality with respect to u_1, u_2 and b_1,

$$|u_1u_2| \geq |u_2b_1| - |b_1u_1| = |u_2b_1| - (|ob_1| - \frac{d}{2})$$

holds. Similarly we have

$$|u_2b_1| \geq |b_1b_2| - |u_2b_2| = |b_1b_2| - (|ob_2| - \frac{d}{2}).$$

Thus we have

$$\begin{aligned} d - |u_1u_2| &\leq d - \{|u_2b_1| - (|ob_1| - \frac{d}{2})\} \\ &\leq \frac{d}{2} + |ob_1| - \{|b_1b_2| - (|ob_2| - \frac{d}{2})\} \\ &= |ob_1| + |ob_2| - |b_1b_2|. \end{aligned} \tag{4.13}$$

In addition, the length of u_1u_2 satisfies the following equation.

$$|u_1u_2| = \frac{d}{2} \cdot 2\sin\frac{\theta}{2} = d\sin\frac{\theta}{2}. \tag{4.14}$$

By (4.12), (4.13) and (4.14),

$$\mathsf{OPT}(P) \geq L + d - |u_1u_2| = L + d - d\sin\frac{\theta}{2}$$

is shown. □

Secondly, we show a better lower bound of L.

Lemma 10. *Let d and θ as defined in* Lemma 9. *Then*

$$L \geq d(\pi - \frac{\theta}{2} + \tan\frac{\theta}{2}). \tag{4.15}$$

Proof. Let C be a circle centered at o with radius $d/2$. From Lemma 7, any polygon edge does not intersect C. Also let endpoints of a visible part of e_{opt} from o be w_1, w_2 in clockwise order around o. Then, we consider two cases: [Case 1] $\angle ow_1w_2 \leq \pi/2$ and $\angle ow_2w_1 \leq \pi/2$ and [Case 2] $\angle ow_1w_2 > \pi/2$ and $\angle ow_2w_1 \leq \pi/2$ (see Figure 4.8, 4.9). Note that the case of $\angle ow_1w_2 \leq \pi/2, \angle ow_2w_1 > \pi/2$ can be treated in a manner similar to Case 2.

[Case 1]: Let w_1^* (resp. w_2^*) be a point on the line segment ow_1 (resp. ow_2) such that w_1w_2 is parallel to $w_1^*w_2^*$ and the line segment $w_1^*w_2^*$ touches the circle C and let h be a tangent point of $w_1^*w_2^*$ and C. Also let $\angle w_1oh = x\theta$ and

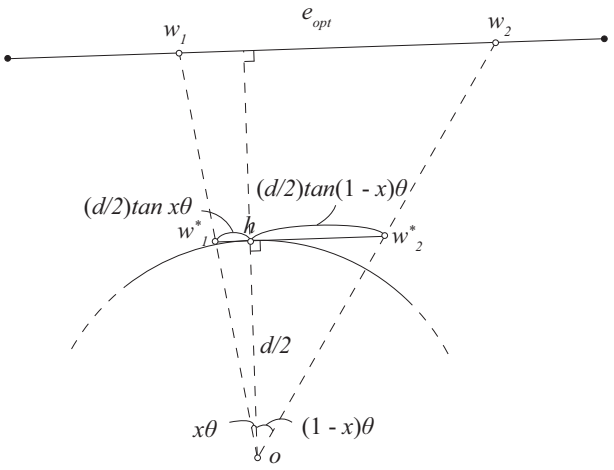

Figure 4.8: Illustration of Case 1 in the proof of Lemma 10

$\angle w_2oh = (1-x)\theta$ with some x $(0 \leq x \leq 1)$. Then the length of $w_1^*w_2^*$ satisfies

$$|w_1^*w_2^*| = \frac{d}{2}\tan x\theta + \frac{d}{2}\tan(1-x)\theta.$$

The right-hand side of this equation attains the minimum value when $x = 1/2$. Thus

$$|w_1^*w_2^*| \geq \frac{d}{2}\tan\frac{\theta}{2} + \frac{d}{2}\tan\frac{\theta}{2} = d\tan\frac{\theta}{2}. \tag{4.16}$$

Furthermore the sum of the visual angle at o formed by a visible part of the boundary other than w_1w_2 is equal to $2\pi - \theta$. Hence we have

$$L \geq \frac{d}{2}(2\pi - \theta) + |w_1w_2|. \tag{4.17}$$

Since $|w_1w_2| \geq |w_1^*w_2^*|$ obviously holds, from (4.16) and (4.17), we obtain

$$L \geq \frac{d}{2}(2\pi - \theta) + d\tan\frac{\theta}{2} = d(\pi - \frac{\theta}{2} + \tan\frac{\theta}{2}).$$

[Case 2]: Let w_1^* (resp. w_2^*) be a point on the line segment ow_1 (resp. ow_2) such that w_1w_2 is parallel to $w_1^*w_2^*$ and $|ow_1^*| = d/2$ (the circumference of C passes through w_1^*). Also let w_2^{**} an intersection point of the line segment ow_2 and the lineperpendicular to the line segment ow_1 through w_1^*. Then

$$|w_1^*w_2^*| > |w_1^*w_2^{**}| = \frac{d}{2}\tan\theta \geq d\tan\frac{\theta}{2}.$$

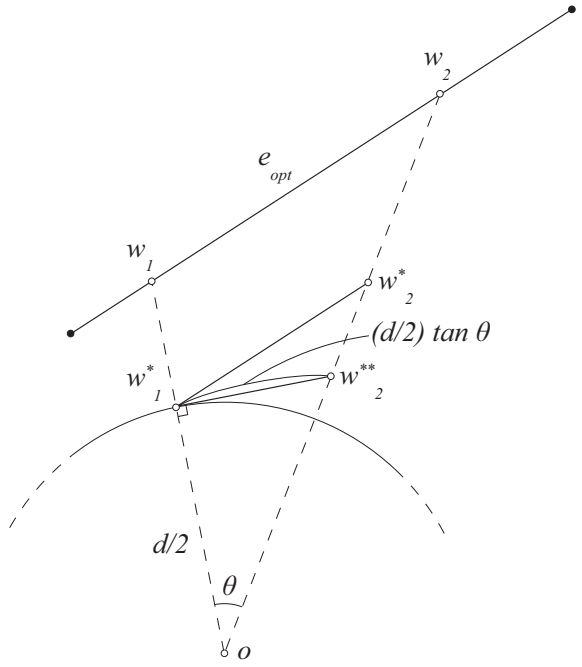

Figure 4.9: Illustration of Case 2 in the proof of Lemma 10

In the same way as Case 1, we obtain $L \geq d(\pi - \theta/2 + \tan(\theta/2))$.

□

By Lemma 9 and 10, we prove the following theorem.

Theorem 6. *The competitive ratio of Algorithm* AOE *is at most* 1.219.

Proof. Let d and θ as defined in Lemma 9. Since $\mathsf{AOE}(P) = L + d$ holds, from (4.10), (4.15), we have

$$\begin{aligned} \frac{\mathsf{AOE}(P)}{\mathsf{OPT}(P)} &\leq \frac{L+d}{L+d-d\sin\frac{\theta}{2}} \\ &\leq \frac{d(\pi - \frac{\theta}{2} + \tan\frac{\theta}{2}) + d}{d(\pi - \frac{\theta}{2} + \tan\frac{\theta}{2}) + d - d\sin\frac{\theta}{2}} \\ &= \frac{\pi - \frac{\theta}{2} + \tan\frac{\theta}{2} + 1}{\pi - \frac{\theta}{2} + \tan\frac{\theta}{2} + 1 - \sin\frac{\theta}{2}} \quad (0 \leq \theta \leq \pi). \end{aligned} \tag{4.18}$$

In the followings, we compute the maximum value of (4.18),

$$\max_{0\leq\theta\leq\pi} \left\{ z(\theta) = \frac{\pi - \frac{\theta}{2} + \tan\frac{\theta}{2} + 1}{\pi - \frac{\theta}{2} + \tan\frac{\theta}{2} + 1 - \sin\frac{\theta}{2}} \right\}. \tag{4.19}$$

Generally the following fact about the fractional program is known [15, 40].

Fact 1. *Let $X \subseteq \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$. Let us consider the following fractional program formulated as*

$$\text{maximize} \left\{ h(x) = \frac{f(x)}{g(x)} \;\middle|\; x \in X \right\}, \tag{4.20}$$

where $g(x) > 0$ is assumed for any $x \in X$. Let $x^* \in \operatorname{argmax}_{x \in X} h(x)$ denote an optimal solution of (4.20) and $\lambda^* = h(x^*)$ denote the optimal value. Furthermore, with a real parameter λ, let $h_\lambda(x) = f(x) - \lambda g(x)$ and $M(\lambda) = \max_{x \in X} h_\lambda(x)$. Then $M(\lambda)$ is monotone decreasing for λ and the followings hold.
(i) $M(\lambda) < 0 \Leftrightarrow \lambda > \lambda^*$, (ii) $M(\lambda) = 0 \Leftrightarrow \lambda = \lambda^*$, (iii) $M(\lambda) > 0 \Leftrightarrow \lambda < \lambda^*$.

In the same way as Fact 1, with a real parameter λ, we define $z_\lambda(\theta)$ and $M(\lambda)$ for $z(\theta)$ as follows.

$$\begin{aligned} z_\lambda(\theta) &= \pi - \frac{\theta}{2} + \tan\frac{\theta}{2} + 1 \\ &\qquad -\lambda(\pi - \frac{\theta}{2} + \tan\frac{\theta}{2} + 1 - \sin\frac{\theta}{2}) \quad (0 \le \theta \le \pi), \\ M(\lambda) &= \max_{0 \le \theta \le \pi} z_\lambda(\theta). \end{aligned}$$

From Fact 1 (ii), λ^* satisfying $M(\lambda^*) = 0$ is equal to (4.19), i.e., the maximum value of $z(\theta)$. Hence we only need to compute λ^*.

Finally, let $\theta^*_\lambda \in \operatorname{argmax}_{0 \le \theta \le \pi} z_\lambda(\theta)$, then we show θ^*_λ is unique. A derivative of $z_\lambda(\theta)$ is calculated as

$$\frac{dz_\lambda}{d\theta} = -\frac{\lambda - 1}{2}\tan^2\frac{\theta}{2} + \frac{\lambda}{2}\cos\frac{\theta}{2}.$$

This derivative is monotone decreasing in the interval $0 \le \theta \le \pi$, therefore $z_\lambda(\theta)$ is concave in this interval, then θ^*_λ is unique. Indeed when $\lambda = 1.219$, $\theta^*_\lambda \simeq 2.0706$ then $M(1.219) \simeq -0.0010 < 0$. Also when $\lambda = 1.218$, $\theta^*_\lambda \simeq 2.0718$ then $M(1.218) \simeq 0.0029 > 0$. Thus we obtain $1.218 < \lambda^* < 1.219$. □

4.3.4 Lower bound analysis

Theorem 7. *The competitive ratio of Algorithm* AOE *is at least* 1.040.

Proof. We consider how Algorithm AOE works for a polygon P_{bad} illustrated in Figure 4.10. We assume that the greater arc from h to c in clockwise order of a circle with radius 10.00 centered at o in the figure is in fact a chain composed of

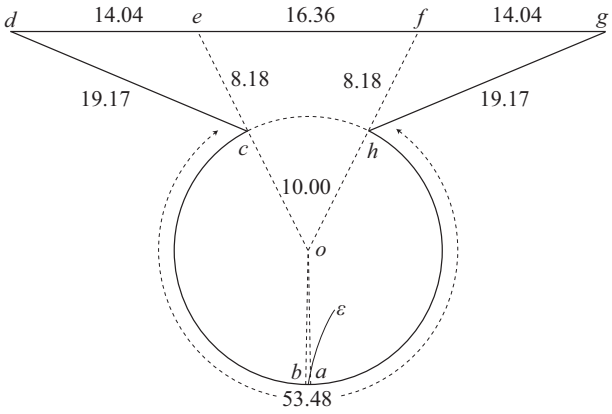

Figure 4.10: Illustration of a polygon P_{bad} in the proof of Theorem 7

sufficiently many small polygon edges of length ϵ. For each small edge s along the arc hc, $inc(o, s) = 20.00 - \epsilon$ holds. The algorithm calculates the increase of a virtual edge (e, f) as $inc(o, (e, f)) \simeq 10.00+8.18+10.00+8.18-16.36 = 20.00$. Comparing these two values, the algorithm chooses a polygon edge (a, b) in the arc hc. Since $L \simeq 136.26$ holds, the tour length of Algorithm AOE for P_{bad} satisfies

$$\mathsf{AOE}(P_{bad}) \simeq 136.26 + 20.00 - \epsilon \geq 156.26 - \epsilon. \tag{4.21}$$

On the other hand, $(d, g) = e_{opt}$ because $inc(o, (d, g)) \simeq 13.89 < 20.00 - \epsilon$ holds. Thus the tour length of an offline optimal algorithm for P_{bad} satisfies

$$\mathsf{OPT}(P_{bad}) \simeq 136.26 + 13.89 \leq 150.16. \tag{4.22}$$

From (4.21) and (4.22), we obtain

$$\frac{\mathsf{AOE}(P_{bad})}{\mathsf{OPT}(P_{bad})} \geq \frac{156.26 - \epsilon}{150.16} \geq 1.0406 - \frac{\epsilon}{150.16}.$$

By letting ϵ be sufficiently small, the theorem follows. □

4.4 Online Exploration Problems in Rectilinear Simple Polygons

In this section, we analyze the competitive ratio of AOE for rectilinear polygons. We are given a rectilinear polygon R consisting of a set of polygon vertices V

and a set of polygon edges E, and the origin $o \in R$. Let R' be the minimum enclosing rectangle of R. Recall that we defined the height of R' as the height of R and also the width of R' as the width of R.

4.4.1 Upper bound analysis

Lemma 11. *For an edge $e^* \in E^*$ satisfying (4.8), let $d = inc(o, e^*)$ and θ $(0 \leq \theta \leq \pi)$ be a visual angle at o formed by a visible part of e_{opt}. Then*

$$L \geq \max\{4d, 2d + 2d \tan\frac{\theta}{2}\}. \tag{4.23}$$

Proof. First, we show $L \geq 4d$. Let C be a circle centered at o with the radius of $d/2$. From Lemma 7, any polygon edge of R does not intersect C. Thus each of the height and width of R is at least d (the diameter of C), namely $L \geq 4d$ holds (see Figure 4.11).

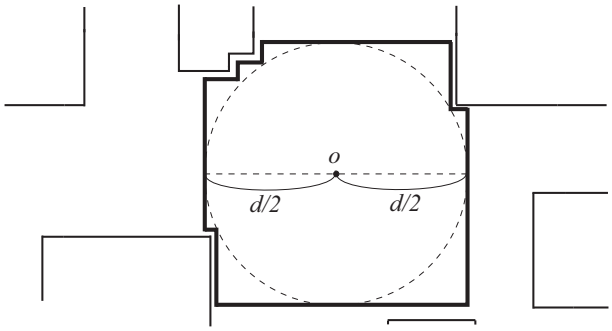

Figure 4.11: Illustration of the minimum enclosing rectilinear polygon of C (represented by thick lines) which is enclosed by R (represented by thin lines)

Secondly, we show $L \geq 2d + 2d\tan(\theta/2)$. Note that we should just consider the case of $4d \leq 2d + 2d\tan(\theta/2)$, namely $\pi/2 \leq \theta \leq \pi$ because $L \geq 4d$ has been proved. Without loss of generality we can assume that e_{opt} is a horizontal edge. We label endpoints of a visible part of e_{opt} from o as w_1, w_2 in clockwise order around o. Let w_1^* (resp. w_2^*) be a point on the line segment ow_1 (resp. ow_2) such that w_1w_2 is parallel to $w_1^*w_2^*$ and the line segment $w_1^*w_2^*$ touches the circle C and h be a tangent point of $w_1^*w_2^*$ and C (see Figure 4.12). Also let $\angle w_1oh = x\theta$ and $\angle w_2oh = (1-x)\theta$ with some x $(0 \leq x \leq 1)$. Then

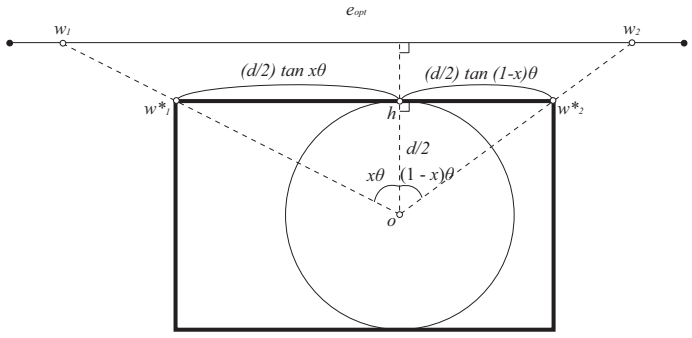

Figure 4.12: Illustration of the minimum enclosing rectangle of C (represented by thick lines) such that θ is more than $\pi/2$

the length of $w_1^* w_2^*$ satisfies

$$\begin{aligned} |w_1^* w_2^*| &= \frac{d}{2}\tan x\theta + \frac{d}{2}\tan(1-x)\theta \\ &\geq \frac{d}{2}\tan\frac{\theta}{2} + \frac{d}{2}\tan\frac{\theta}{2} = d\tan\frac{\theta}{2}. \end{aligned}$$

Thus the width of R is at least $d\tan(\theta/2)$ and the height of R is at least d, then $L \geq 2d + 2d\tan(\theta/2)$ holds. □

Theorem 8. *For a rectilinear polygon, the competitive ratio of Algorithm* AOE *is at most* 1.167.

Proof. Based on (4.23), we consider two cases; (Case 1) $0 \leq \theta < \pi/2$ and (Case 2) $\pi/2 \leq \theta \leq \pi$. Note that $4d > 2d + 2d\tan(\theta/2)$ holds in Case 1 and $4d \leq 2d + 2d\tan(\theta/2)$ holds in the other.
Case 1: From $L \geq 4d$ and (4.10), we obtain

$$\begin{aligned} \frac{\mathsf{AOE}(R)}{\mathsf{OPT}(R)} &\leq \frac{4d+d}{4d+d-d\sin\frac{\theta}{2}} = \frac{5}{5-\sin\frac{\theta}{2}} \\ &< \frac{5}{5-\sin\frac{\pi}{4}} \leq 1.165. \end{aligned}$$

Case 2: From $L \geq 2d + 2d\tan(\theta/2)$ and (4.10), we obtain

$$\begin{aligned} \frac{\mathsf{AOE}(R)}{\mathsf{OPT}(R)} &\leq \frac{2d+2d\tan\frac{\theta}{2}+d}{2d+2d\tan\frac{\theta}{2}+d-d\sin\frac{\theta}{2}} \\ &= \frac{3+2\tan\frac{\theta}{2}}{3+2\tan\frac{\theta}{2}-\sin\frac{\theta}{2}}. \end{aligned} \tag{4.24}$$

We will compute the maximum value of (4.24) as in the proof of Theorem 6

by defining $z_\lambda(\theta)$ and $M(\lambda)$ for a real parameter λ as follows.

$$\begin{aligned} z_\lambda(\theta) &= 3 + 2\tan\frac{\theta}{2} \\ &\quad -\lambda(3 + 2\tan\frac{\theta}{2} - \sin\frac{\theta}{2}) \quad (\frac{\pi}{2} \le \theta \le \pi) \\ M(\lambda) &= \max_{\frac{\pi}{2}\le\theta\le\pi} z_\lambda(\theta) \end{aligned}$$

Let $\theta^*_\lambda \in \mathrm{argmax}_{0\le\theta\le\pi}\, z_\lambda(\theta)$, then a derivative of $z_\lambda(\theta)$ is calculated as

$$\frac{dz_\lambda}{d\theta} = -(\lambda - 1)\frac{1}{\cos^2\frac{\theta}{2}} + \frac{\lambda}{2}\cos\frac{\theta}{2}.$$

This derivative is monotone decreasing in the interval $\pi/2 \le \theta \le \pi$, therefore $z_\lambda(\theta)$ is concave in this interval, then θ^*_λ is unique. Indeed when $\lambda = 1.167$, $\theta^*_\lambda \simeq 1.7026$ then $M(1.167) \simeq -0.0044 < 0$. Also when $\lambda = 1.166$, $\theta^*_\lambda \simeq 1.7056$ then $M(1.166) \simeq 7.6 \times 10^{-5} > 0$. Thus we obtain $1.166 < \lambda^* < 1.167$. □

4.4.2 Lower bound analysis

Theorem 9. *The competitive ratio of Algorithm* AOE *for a rectilinear polygon is at least* 1.034.

Proof. We consider how Algorithm AOE works for a polygon R_{bad} illustrated in Figure 4.13. Let $\widetilde{mf}$ denote the polygonal chain from m to f in clockwise order

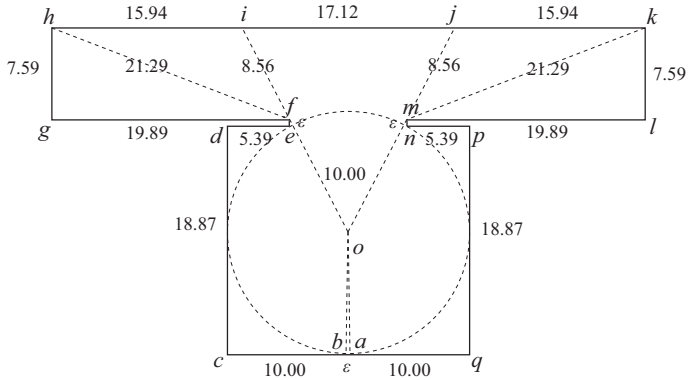

Figure 4.13: Illustration of a rectilinear polygon R_{bad} in the proof of Theorem 9

around o composed of segments mn, np, pq, qc, cd, de and ef in the figure. We assume that $\widetilde{mf}$ is in fact a chain composed of sufficiently many small polygon edges of length ϵ. Notice that segments mn and ef are edges of length ϵ. Also

we can assume that ab is a polygon edge in the middle of q and c such that $|oa| = |ob| \simeq 10.00$. Then $inc(o,(a,b)) \simeq 20.00 - \epsilon$ and $inc(o,s) \geq 20.00 - \epsilon$ holds for each small edge s along $\widetilde{mf}$. The algorithm calculates the increase of a virtual edge (i,j) as $inc(o,(i,j)) \simeq 10.00+8.56+10.00+8.56-17.12 = 20.00$. Thus the algorithm chooses the polygon edge (a,b). Since $L \simeq 172.48$ holds, the tour length of Algorithm AOE for R_{bad} satisfies

$$\mathsf{AOE}(R_{bad}) \simeq 172.48 + 20.00 - \epsilon \geq 192.48 - \epsilon. \tag{4.25}$$

On the other hand, $(h,k) = e_{opt}$ because $inc(o,(h,k)) \simeq 13.58 < 20.00 - \epsilon$ holds. Thus the tour length of an offline optimal algorithm for R_{bad} satisfies

$$\mathsf{OPT}(R_{bad}) \simeq 172.48 + 13.58 \leq 186.07. \tag{4.26}$$

From (4.25) and (4.26), we obtain

$$\frac{\mathsf{AOE}(R_{bad})}{\mathsf{OPT}(R_{bad})} \geq \frac{192.48 - \epsilon}{186.07} \geq 1.0344 - \frac{\epsilon}{186.07}.$$

By letting ϵ be sufficiently small, the theorem follows. □

4.5 Conclusion

In Section 4.3, we considered the online exploration problem in simple polygons by a single searcher, and proposed an online algorithm which achieves a competitive ratio at most 1.219 and at least 1.040. In Section 4.4, we considered the online exploration problem in rectilinear simple polygons by a single searcher, and showed that the same algorithm as in Section 4.3 achieves a competitive ratio at most 1.167 and at least 1.034 restricting the class of object space to rectilinear simple polygons.

Part II

Sink Location Problems in Dynamic Networks under Incomplete Information

Chapter 5

Minimax Regret Sink Location Problems in Dynamic Path Networks

5.1 Introduction

In this chapter, we consider the minimax regret sink location problems in dynamic path networks with uniform capacity. Here, we assume the continuous model. A dynamic path network represents a single road with multiple lanes in reality. In this chapter, we will develop two polynomial algorithms which can solve the 1-sink location problem and k-sink location problem, respectively.

5.2 Outline

In Section 5.3, we consider the minimax regret 1-sink location problem in dynamic path networks with uniform capacity and propose an algorithm for the problem. In Section 5.4, we consider the optimal k-sink location problem in dynamic path networks with uniform capacity and propose an algorithm for the problem in order to develop an algorithm for the minimax regret k-sink location problem. Also, in Section 5.4, we consider the optimal k-sink location problem in a dynamic path network with general capacities and show that the problem can be solved in polynomial time. In Section 5.5, we consider the minimax regret k-sink location problem in dynamic path networks with uniform capacity and propose an algorithm for the problem, which uses the algorithm proposed in Section 5.4 as a subroutine.

5.3 Minimax Regret 1-Sink Location Problem

In this section, we consider the minimax regret 1-sink location problem in dynamic path networks with uniform capacity. We are given a dynamic path network with uniform capacity under uncertain supplies $\mathcal{N} = (P = (V, E), W, l, c', \tau)$. Let $\mathcal{S}$ denote a set of scenarios (refer (2.6)). Referring (2.7), the evacuation time of a sink location $x \in P$ under a scenario $s \in \mathcal{S}$ is represented as $\Theta^s(\boldsymbol{x}, \mathcal{P})$ where $\boldsymbol{x} = \{x\}$ and $\mathcal{P} = \{V\}$. For simplicity, in this section, we use the notation $\Theta^s(x)$ to denote the evacuation time of a sink location x under a scenario s. For a sink location $x \in P$ and a scenario $s \in \mathcal{S}$, let $\Theta^s_L(x)$ (resp. $\Theta^s_R(x)$) denote the minimum time required to send all supplies under the scenario s on the part of P consisting of all points $p \in P$ such that $p < x$ (resp. $x < p$) to x. Then, by (2.12), $\Theta^s(x)$ can be represented as follows:

$$\Theta^s(x) = \max\left\{\Theta^s_L(x), \Theta^s_R(x)\right\}. \tag{5.1}$$

Also, by (2.17) and (2.18), we have the following formulae for continuous model assuming that the uniform capacity of edge is set as $c' = 1$:

$$\Theta^s_L(x) = \max_i \left\{ (x - v_i)\tau + \sum_{1 \le j \le i} w^s(v_j) \;\middle|\; v_i \in [v_1, x) \right\}, \tag{5.2}$$

$$\Theta^s_R(x) = \max_i \left\{ (v_i - x)\tau + \sum_{i \le j \le n} w^s(v_j) \;\middle|\; v_i \in (x, v_n] \right\}. \tag{5.3}$$

Let $f^{s(i)}_L(x)$ and $f^{s(i)}_R(x)$ denote functions defined as follows: for an integer i with $1 \le i \le n-1$,

$$f^{s(i)}_L(x) = (x - v_i)\tau + \sum_{1 \le j \le i} w^s(v_j) \ (x > v_i), \tag{5.4}$$

and for an integer i with $2 \le i \le n$,

$$f^{s(i)}_R(x) = (v_i - x)\tau + \sum_{i \le j \le n} w^s(v_j) \ (x < v_i). \tag{5.5}$$

Then, $\Theta_L^s(x)$ and $\Theta_R^s(x)$ are expressed as follows:

$$\Theta_L^s(x) = \max_i \left\{ f_L^{s(i)}(x) \;\middle|\; v_i \in [v_1, x) \right\}, \tag{5.6}$$

$$\Theta_R^s(x) = \max_i \left\{ f_R^{s(i)}(x) \;\middle|\; v_i \in (x, v_n] \right\}. \tag{5.7}$$

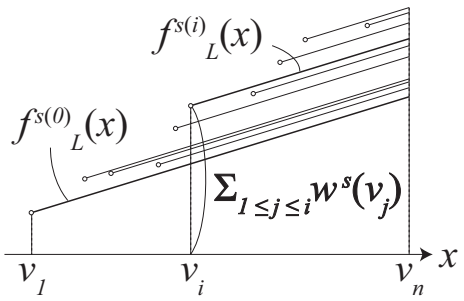

Figure 5.1: Functions $f_L^{s(i)}(x)$ for $1 \le i \le n-1$

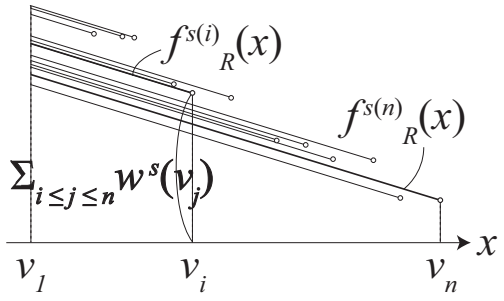

Figure 5.2: Functions $f_R^{s(i)}(x)$ for $2 \le i \le n$

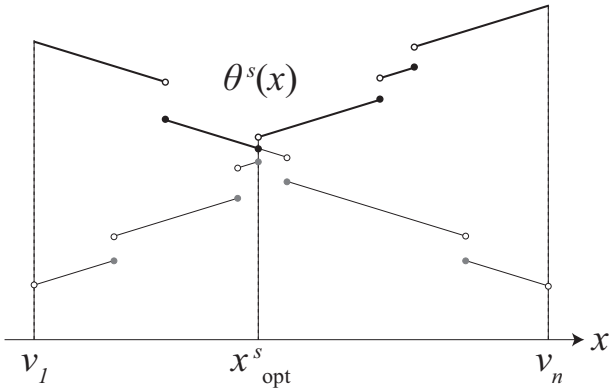

Figure 5.3: A function $\Theta^s(x)$

A function $f_L^{s(i)}(x)$ is drawn as a left-open segment with a positive slope τ starting from $(v_i, \sum_{1 \le j \le i} w^s(v_j))$ and ending at $(v_n, (v_n - v_i)\tau + \sum_{1 \le j \le i} w^s(v_j))$ (see Figure 5.1) while $f_R^{s(i)}(x)$ is drawn as a right-open segment with a negative slope $-\tau$ starting from $(v_1, (v_i - v_1)\tau + \sum_{i \le j \le n} w^s(v_j))$ and ending at $(v_i, \sum_{i \le j \le n} w^s(v_j))$ (see Figure 5.2). Thus $\Theta_L^s(x)$ is the upper envelope of these $n-1$ segments, and so $\Theta_L^s(x)$ is a monotonically increasing function of x. Symmetrically, $\Theta_R^s(x)$ is a monotonically decreasing function of x. Therefore, $\Theta^s(x)$ is a unimodal function, so there is a unique point in P which minimizes $\Theta^s(x)$ (see Figure 5.3). In the following, let x^s_{opt} denote such a point

in P:

$$x^s_{\text{opt}} = \text{argmin}\{\Theta^s(x) \mid v_1 \leq x \leq v_n\}. \tag{5.8}$$

We have the following propositions.

Proposition 1. *Under a scenario $s \in \mathcal{S}$,*
(i) *x^s_{opt} is unique,*
(ii) *for a sink location $x < x^s_{\text{opt}}$, $\Theta^s_L(x) < \Theta^s_R(x)$ holds, and*
(iii) *for a sink location $x > x^s_{\text{opt}}$, $\Theta^s_L(x) > \Theta^s_R(x)$ holds.*

Note that Proposition 1(ii)(iii) implies that $\Theta^s(x) = \Theta^s_R(x)$ holds for $x < x^s_{\text{opt}}$ and $\Theta^s(x) = \Theta^s_L(x)$ holds for $x > x^s_{\text{opt}}$.

For a given scenario $s \in \mathcal{S}$, the optimal evacuation time Θ^s_{opt} is represented as follows (refer (2.8)):

$$\Theta^s_{\text{opt}} = \min\{\Theta^s(x) \mid x \in P\}. \tag{5.9}$$

Referring (2.9), the regret of a sink location $x \in P$ under a scenario $s \in \mathcal{S}$ is represented as $R^s(\boldsymbol{x}, \mathcal{P})$ where $\boldsymbol{x} = \{x\}$ and $\mathcal{P} = \{V\}$. For simplicity, in this section, we use the notation $R^s(x)$ to denote the regret of a sink location x under a scenario s. Then, for given a scenario $s \in \mathcal{S}$ and a sink location $x \in P$, the regret $R^s(x)$ is represented as follows:

$$R^s(x) = \Theta^s(x) - \Theta^s_{\text{opt}}. \tag{5.10}$$

We also use the notation $R_{\max}(x)$ to denote the maximum regret of a sink location x instead of $R_{\max}(\boldsymbol{x}, \mathcal{P})$ (refer (2.10)). Then, for a given sink location $x \in P$, the maximum regret $R_{\max}(x)$ is represented as follows:

$$R_{\max}(x) = \max_{s \in \mathcal{S}} R^s(x). \tag{5.11}$$

We call a scenario which maximizes $R^s(x)$ a worst case scenario (refer 2.11).

The goal is to find the minimax regret sink location x^* which minimizes $R_{\max}(x)$. Therefore, the minimax regret 1-sink location problem in a path P is defined as follows:

$$\text{minimize } \{R_{\max}(x) \mid x \in P\}. \tag{5.12}$$

5.3.1 Properties

In this subsection, we show some key properties of the problem. First, let us recall the following claim.

Claim 3. *For a scenario $s \in \mathcal{S}$, a function $\Theta^s(x)$ is unimodal in x.*

In contraposition to Proposition 1(ii)(iii), we also have the following claim.

Claim 4. *For a scenario $s \in \mathcal{S}$ and a sink location $x \in P$,*
(i) *if $\Theta^s_L(x) \geq \Theta^s_R(x)$ holds, $x^s_{\mathrm{opt}} \leq x$ holds, and*
(ii) *if $\Theta^s_L(x) \leq \Theta^s_R(x)$ holds, $x^s_{\mathrm{opt}} \geq x$ holds.*

For a given scenario $s \in \mathcal{S}$, by definition of (5.10) and Claim 3, a function $R^s(x)$ is also unimodal in x. Thus, a function $R_{\max}(x)$ is unimodal in x since it is the upper envelope of unimodal functions by (5.11).

Claim 5. *A function $R_{\max}(x)$ is unimodal in x.*

In the following, let x^* denote the minimax regret sink location in P. Then, we obtain the following lemma.

Lemma 12. *For a sink location $x \in P$, let $\hat{s} = \mathrm{argmax}\{R^s(x) \mid s \in \mathcal{S}\}$. Then,*
(i) *if $\Theta^{\hat{s}}_L(x) \geq \Theta^{\hat{s}}_R(x)$ holds, $x^* \leq x$ holds, and*
(ii) *if $\Theta^{\hat{s}}_L(x) \leq \Theta^{\hat{s}}_R(x)$ holds, $x^* \geq x$ holds.*

Proof. We only prove (i) by contradiction: suppose that $x^* > x$ and $\Theta^{\hat{s}}_L(x) \geq \Theta^{\hat{s}}_R(x)$ hold. By Claim 4(i), $x^{\hat{s}}_{\mathrm{opt}} \leq x$ holds. Then, $\Theta^{\hat{s}}(x^*) > \Theta^{\hat{s}}(x)$ holds by Claim 3, thus $R^{\hat{s}}(x^*) > R^{\hat{s}}(x)$ also holds by (5.10). We have $R_{\max}(x^*) \geq R^{\hat{s}}(x^*)$ by the maximality of $R_{\max}(x^*)$, and $R^{\hat{s}}(x) = R_{\max}(x)$ by definition of $\hat{s}$. Thus, $R_{\max}(x^*) > R_{\max}(x)$ holds, which contradicts the optimality of x^*. □

For a scenario $s \in \mathcal{S}$ and an integer i with $1 \leq i \leq n$, let s^i_+ denote a scenario such that

$$w^{s^i_+}(v_i) = w^+(v_i) \text{ and } w^{s^i_+}(v_j) = w^s(v_j) \text{ for } j \neq i,$$

and s^i_- denote a scenario such that

$$w^{s^i_-}(v_i) = w^-(v_i) \text{ and } w^{s^i_-}(v_j) = w^s(v_j) \text{ for } j \neq i.$$

By (5.4), $f^{s(i)}_L(x)$ is defined on $x > v_i$ for i with $1 \leq i \leq n-1$. Thus, for a sink location x such that $v_i < x \leq v_n$, $f^{s(i)}_L(x) \leq f^{s^i_+(i)}_L(x)$ and $f^{s^i_-(i)}_L(x) \leq f^{s(i)}_L(x)$ always hold for any i. Moreover, by these facts and (5.6), we also have $\Theta^s_L(x) \leq \Theta^{s^i_+}_L(x)$ and $\Theta^{s^i_-}_L(x) \leq \Theta^s_L(x)$. We have the following claim.

Claim 6. *For a scenario $s \in \mathcal{S}$, a sink location $x \in P$ and an integer i with $1 \leq i \leq n$ such that $v_1 \leq v_i \leq x$* (resp. $x \leq v_i \leq v_n$),
(i) $\Theta_L^s(x) \leq \Theta_L^{s_+^i}(x)$ (resp. $\Theta_R^s(x) \leq \Theta_R^{s_+^i}(x)$), *and*
(ii) $\Theta_L^{s_-^i}(x) \leq \Theta_L^s(x)$ (resp. $\Theta_R^{s_-^i}(x) \leq \Theta_R^s(x)$) *holds.*

A scenario $s \in \mathcal{S}$ is said to be *left-dominant* (resp. *right-dominant*) if for a given i with $1 \leq i \leq n$, $w^s(v_j) = w^+(v_j)$ for $1 \leq j \leq i$ and $w^s(v_j) = w^-(v_j)$ for $i+1 \leq j \leq n$ hold (resp. $w^s(v_j) = w^-(v_j)$ for $1 \leq j \leq i$ and $w^s(v_j) = w^+(v_j)$ for $i+1 \leq j \leq n$ hold). Let $\mathcal{S}_L$ (resp. $\mathcal{S}_R$) denote the set of all left-dominant (resp. right-dominant) scenarios. $\mathcal{S}_L$ consists of the following $n+1$ scenarios:

$$\begin{aligned} &s_L^i = (w^+(v_1), \ldots, w^+(v_i), w^-(v_{i+1}), \ldots, w^-(v_n)) \text{ for } 1 \leq i \leq n-1, \text{ and} \\ &s_L^n = (w^+(v_1), w^+(v_1), \ldots, w^+(v_n)), \end{aligned} \tag{5.13}$$

and $\mathcal{S}_R$ consists of the following $n+1$ scenarios:

$$\begin{aligned} &s_R^i = (w^-(v_1), \ldots, w^-(v_i), w^+(v_{i+1}), \ldots, w^+(v_n)) \text{ for } 1 \leq i \leq n-1, \text{ and} \\ &s_R^n = (w^-(v_1), w^-(v_1), \ldots, w^-(v_n)). \end{aligned} \tag{5.14}$$

The following is a key lemma.

Lemma 13. *For a sink location $x \in P$, there exists a worst case scenario which belongs to $\mathcal{S}_L \cup \mathcal{S}_R$.*

Proof. Let $\hat{s} = \text{argmax}\{R^s(x) \mid s \in \mathcal{S}\}$. Here, we only prove for a sink location x such that $\Theta_L^{\hat{s}}(x) \geq \Theta_R^{\hat{s}}(x)$. Suppose that $v_{k-1} < x \leq v_k$ with an integer k with $2 \leq k \leq n$ and $l = \text{argmax}\{f_L^{\hat{s}(i)}(x) \mid 1 \leq i \leq k-1\}$, i.e.,

$$\Theta^{\hat{s}}(x) = f_L^{\hat{s}(l)}(x) = (x - v_l)\tau + \sum_{1 \leq j \leq l} w^{\hat{s}}(v_j). \tag{5.15}$$

Then, we actually prove that $R^{s_L^l}(x) \geq R^{\hat{s}}(x)$ holds. If $\hat{s}$ is not equal to s_L^l, we have two cases: [Case 1] there exists an integer i with $1 \leq i \leq l$ such that $w^{\hat{s}}(v_i) < w^+(v_i)$, and [Case 2] there exists an integer i with $l+1 \leq i \leq n$ such that $w^{\hat{s}}(v_i) > w^-(v_i)$. If we can show that $R^{\hat{s}_+^i}(x) \geq R^{\hat{s}}(x)$ holds for Case 1 and $R^{\hat{s}_-^i}(x) \geq R^{\hat{s}}(x)$ holds for Case 2, we will eventually obtain $R^{s_L^l}(x) \geq R^{\hat{s}}(x)$ by repeatedly applying the same discussion as long as there exists such a vertex v_i.

[Case 1]: Let $\Delta = w^+(v_i) - w^{\hat{s}}(v_i)$. We first notice

$$\begin{aligned}\Theta^{\hat{s}^i_+}(x) &= \Theta_L^{\hat{s}^i_+}(x), \quad \text{and} \\ \Theta_L^{\hat{s}^i_+}(x) &= (x - v_l)\tau + \sum_{1\le j\le l} w^{\hat{s}^i_+}(v_j) = \Theta^{\hat{s}}(x) + \Delta\end{aligned}$$

by (5.2) and (5.15). Thus, we have

$$\Theta^{\hat{s}^i_+}(x) = \Theta^{\hat{s}}(x) + \Delta. \tag{5.16}$$

By the optimality of $x^{\hat{s}^i_+}_{\text{opt}}$ under $\hat{s}^i_+$ (see (5.8)), $\Theta^{\hat{s}^i_+}_{\text{opt}} \le \Theta(x^{\hat{s}}_{\text{opt}}, \hat{s}^i_+)$ holds. Here, we show

$$\Theta^{\hat{s}^i_+}(x^{\hat{s}}_{\text{opt}}) \le \Theta^{\hat{s}}(x^{\hat{s}}_{\text{opt}}) + \Delta = \Theta^{\hat{s}}_{\text{opt}} + \Delta \tag{5.17}$$

for the subcase of $x^{\hat{s}}_{\text{opt}} > v_i$ (the other case can be similarly treated). In this case, $\Theta_L^{\hat{s}^i_+}(x^{\hat{s}}_{\text{opt}}) \le \Theta^{\hat{s}}_L(x^{\hat{s}}_{\text{opt}}) + \Delta$ (see Figure 5.4) and $\Theta_R^{\hat{s}^i_+}(x^{\hat{s}}_{\text{opt}}) = \Theta^{\hat{s}}_R(x^{\hat{s}}_{\text{opt}})$ hold by the definitions of (5.2) and (5.3), which implies that (5.17) holds. Thus, we

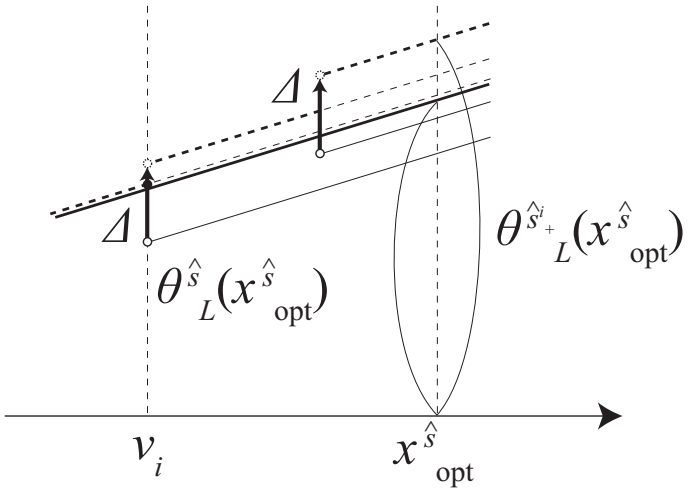

Figure 5.4: Subcase of Case 1: $x^{\hat{s}}_{\text{opt}} > v_i$

have

$$\Theta^{\hat{s}^i_+}_{\text{opt}} \le \Theta^{\hat{s}}_{\text{opt}} + \Delta. \tag{5.18}$$

By (5.10), (5.16) and (5.18), we obtain $R^{\hat{s}^i_+}(x) \ge R^{\hat{s}}(x)$.
[Case 2]: In this case, $\Theta^{\hat{s}^i_-}(x) = \Theta_L^{\hat{s}^i_-}(x)$ and $\Theta_L^{\hat{s}^i_-}(x) = \Theta^{\hat{s}}_L(x)$ by (5.2) and

(5.15). Thus, we have

$$\Theta^{\hat{s}^i_-}(x) = \Theta^{\hat{s}}(x). \tag{5.19}$$

By the optimality of $x^{\hat{s}^i_-}_{\rm opt}$ under $\hat{s}^i_-$, $\Theta^{\hat{s}^i_-}_{\rm opt} \leq \Theta^{\hat{s}^i_-}(x^{\hat{s}}_{\rm opt})$ holds. By Claim 6(ii) and (5.9), $\Theta^{\hat{s}^i_-}(x^{\hat{s}}_{\rm opt}) \leq \Theta^{\hat{s}}(x^{\hat{s}}_{\rm opt}) = \Theta^{\hat{s}}_{\rm opt}$ holds. Thus, we have

$$\Theta^{\hat{s}^i_-}_{\rm opt} \leq \Theta^{\hat{s}}_{\rm opt}. \tag{5.20}$$

By (5.10), (5.19) and (5.20), we obtain $R^{\hat{s}^i_-}(x) \geq R^{\hat{s}}(x)$. □

We also have the following lemma.

Lemma 14. *For a scenario $s \in \mathcal{S}$ and an integer i with $1 \leq i \leq n$ such that $v_1 \leq v_i \leq x^s_{\rm opt}$ (resp. $x^s_{\rm opt} \leq v_i \leq v_n$), $v_i \leq x^{s^i_+}_{\rm opt} \leq x^s_{\rm opt}$ (resp. $x^s_{\rm opt} \leq x^{s^i_+}_{\rm opt} \leq v_i$) holds.*

Proof. We only prove $v_i \leq x^{s^i_+}_{\rm opt} \leq x^s_{\rm opt}$ for a given integer i with $1 \leq i \leq n$ such that $v_1 \leq v_i \leq x^s_{\rm opt}$ (the other case can be similarly treated). We first prove $x^{s^i_+}_{\rm opt} \leq x^s_{\rm opt}$ by contradiction: suppose that $x^{s^i_+}_{\rm opt} > x^s_{\rm opt}$. Let $x_{\rm mid}$ be the midpoint of $x^{s^i_+}_{\rm opt}$ and $x^s_{\rm opt}$:

$$x_{\rm mid} = \frac{x^{s^i_+}_{\rm opt} + x^s_{\rm opt}}{2}.$$

Then, by Proposition 1(ii)(iii), we have

$$\Theta^{s^i_+}_L(x_{\rm mid}) < \Theta^{s^i_+}_R(x_{\rm mid}) \text{ and } \Theta^s_R(x_{\rm mid}) < \Theta^s_L(x_{\rm mid}). \tag{5.21}$$

Note that by $x^s_{\rm opt} < x_{\rm mid}$ and the assumption of $v_i \leq x^s_{\rm opt}$, we have $v_i < x_{\rm mid}$. By (5.7), $\Theta^s_R(x_{\rm mid})$ is the maximum of $f^{s(j)}_R(x_{\rm mid})$ for all j such that $v_j > x_{\rm mid}$, so $\Theta^s_R(x_{\rm mid})$ does not change if $w^s(v_i)$ increases to $w^+(v_i)$, i.e.,

$$\Theta^{s^i_+}_R(x_{\rm mid}) = \Theta^s_R(x_{\rm mid}). \tag{5.22}$$

Thus, by (5.21) and (5.22), we obtain $\Theta^{s^i_+}_L(x_{\rm mid}) < \Theta^s_L(x_{\rm mid})$ which contradicts Claim 6(i).

We next prove $v_i \leq x^{s^i_+}_{\rm opt}$. If $\Theta^{s^i_+}_L(v_i) \leq \Theta^{s^i_+}_R(v_i)$ holds, $v_i \leq x^{s^i_+}_{\rm opt}$ by Claim 4. We now consider the case of $\Theta^{s^i_+}_L(v_i) > \Theta^{s^i_+}_R(v_i)$. By (5.6) and (5.7),

$$\Theta^{s^i_+}_L(v_i) = \Theta^s_L(v_i) \text{ and } \Theta^{s^i_+}_R(v_i) = \Theta^s_R(v_i)$$

hold, i.e.,

$$\Theta^{s^i_+}(v_i) \;=\; \Theta^s(v_i). \tag{5.23}$$

Thus, we can derive $\Theta^s_L(v_i) > \Theta^s_R(v_i)$ from $\Theta^{s^i_+}_L(v_i) > \Theta^{s^i_+}_R(v_i)$, which implies $v_i \geq x^s_{\rm opt}$ by Claim 4. By this and the condition of $v_i \leq x^s_{\rm opt}$, we have

$$x^s_{\rm opt} = v_i. \tag{5.24}$$

Here, we show that $x^{s^i_+}_{\rm opt} = v_i$ also holds. Suppose otherwise, i.e., $x^{s^i_+}_{\rm opt} \neq v_i$. Then, we have

$$\Theta^s(x^{s^i_+}_{\rm opt}) \;\leq\; \Theta^{s^i_+}(x^{s^i_+}_{\rm opt}), \quad \text{and} \tag{5.25}$$

$$\Theta^{s^i_+}(x^{s^i_+}_{\rm opt}) \;<\; \Theta^{s^i_+}(v_i) \tag{5.26}$$

by Claim 6(i) and the optimality of $x^{s^i_+}_{\rm opt}$, respectively. From (5.23), (5.25), (5.26) and (5.24), we can derive $\Theta^s(x^{s^i_+}_{\rm opt}) < \Theta^s(x^s_{\rm opt})$, which contradicts the optimality of $x^s_{\rm opt}$ under s. Therefore, $x^{s^i_+}_{\rm opt} = v_i$ holds if $\Theta^{s^i_+}_L(v_i) > \Theta^{s^i_+}_R(v_i)$, which implies that $v_i \leq x^{s^i_+}_{\rm opt}$ always holds. □

5.3.2 Algorithm

We will show an $O(n \log n)$ time algorithm which computes a minimax regret sink location x^* minimizing a function $R_{\max}(x)$. The algorithm consists of the following two phases.

Phase 1: Compute $\Theta^{s^i_L}_{\rm opt}$ and $\Theta^{s^i_R}_{\rm opt}$ for all i with $1 \leq i \leq n$.

Phase 2: Compute a minimax regret sink location x^*.

In Phase 2, the algorithm applies binary search to find x^* by the unimodality of $R_{\max}(x)$ (recall Claim 5). In order to apply binary search, the algorithm needs to compute $R_{\max}(v_j)$ for any j with $1 \leq j \leq n$ by computing the maximum of $\Theta^s(v_j) - \Theta^s_{\rm opt}$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$. For this purpose, the algorithm prepares $\Theta^s_{\rm opt}$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$ in Phase 1. In the following, we will explain Phases 1 and 2 in detail, respectively.

Phase 1

In this section, we show how to compute $\Theta^{s^i_L}_{\rm opt}$ for all i with $1 \leq i \leq n$. Computing $\Theta^{s^i_R}_{\rm opt}$ can be done similarly, and thus is omitted. Now, let us

consider how to compute $\Theta^{s_L^i}_{\mathrm{opt}}$ for a given i with $1 \leq i \leq n$. If $\Theta^{s_L^i}_L(v_j)$ and $\Theta^{s_L^i}_R(v_j)$ are computed for some j with $1 \leq j \leq n$, the algorithm can determine whether $\Theta^{s_L^i}_L(v_j) \geq \Theta^{s_L^i}_R(v_j)$ or $\Theta^{s_L^i}_L(v_j) \leq \Theta^{s_L^i}_R(v_j)$, which implies by Claim 4 $x^{s_L^i}_{\mathrm{opt}} \leq v_j$ or $x^{s_L^i}_{\mathrm{opt}} \geq v_j$, respectively. Therefore, the algorithm can apply binary search to find adjacent two vertices v_{l-1} and v_l with some l with $2 \leq l \leq n$ such that $\Theta^{s_L^i}_L(v_{l-1}) \leq \Theta^{s_L^i}_R(v_{l-1})$ and $\Theta^{s_L^i}_L(v_l) \geq \Theta^{s_L^i}_R(v_l)$, i.e., $x^{s_L^i}_{\mathrm{opt}}$ exists in the interval $[v_{l-1}, v_l]$. We will explain the details about how to compute $x^{s_L^i}_{\mathrm{opt}}$ later.

When $x^{s_L^i}_{\mathrm{opt}}$ is found, $\Theta^{s_L^i}_{\mathrm{opt}}$ is immediately computed by the definition of $\Theta^{s_L^i}_{\mathrm{opt}} = \Theta^{s_L^i}(x^{s_L^i}_{\mathrm{opt}})$. Thus, in order to compute $\Theta^{s_L^i}_{\mathrm{opt}}$ for a given i with $1 \leq i \leq n$, the algorithm needs to efficiently compute $\Theta^{s_L^i}_L(v_j)$ and $\Theta^{s_L^i}_R(v_j)$ for any j with $1 \leq j \leq n$. For this purpose, the algorithm constructs two data structures T_L and T_R. We first show that T_L and T_R can be constructed in $O(n \log n)$ time and $\Theta^{s_L^i}_L(v_j)$ or $\Theta^{s_L^i}_R(v_j)$ for any i and j with $1 \leq i \leq n$ and $1 \leq j \leq n$ can be computed in $O(\log n)$ time by using T_L and T_R, respectively. Here, we explain T_L in detail (T_R is can be constructed in a symmetric manner). Let us consider a priority search tree with $n+1$ leaves $l_0, l_1, \ldots, l_n$ corresponding to vertices $v_0, v_1, \ldots, v_n$ and internal nodes such that each internal node has pointers to left and right children. For a node ν in a priority search tree, let $\kappa_L(\nu)$ (resp. $\kappa_R(\nu)$) denote the left (resp. right) child of ν, and $i_{\min}(\nu)$ (resp. $i_{\max}(\nu)$) denote the index of a minimum (resp. maximum) leaf of a subtree rooted at ν, which are stored at ν. Note that for a leaf l_j, $i_{\min}(l_j) = i_{\max}(l_j) = j$ holds. For i with $1 \leq i \leq n$, let T^i_L denote a priority search tree such that each node (including leaf) ν stores indices $i_{\min}(\nu)$ and $i_{\max}(\nu)$,

$$value(\nu; i) = \max\Big\{ -v_k\tau + \sum_{1 \leq l \leq k} w^{s_L^i}(v_l) \;\Big|\; i_{\min}(\nu) \leq k \leq i_{\max}(\nu) \Big\} \tag{5.27}$$

and the corresponding index of leaf that attains the maximum. Then, for any vertex v_j with j with $1 \leq j \leq n$, by (5.4) and (5.27), we have

$$v_j\tau + value(\nu; i) = \max\{ f_L^{s_L^i(k)}(v_j) \mid i_{\min}(\nu) \leq k \leq i_{\max}(\nu) \}. \tag{5.28}$$

Also, for i with $1 \leq i \leq n$, let Π^i_L be a data structure along the path in T^i_L from leaf l_i to the root (see Figure 5.5). Instead of keeping the whole tree T^i_L for i with $2 \leq i \leq n$, we only store Π^i_L. This is enough for our purpose. Thus, T_L basically consists of a priority search tree T^1_L and path data structures Π^i_L for i with $2 \leq i \leq n$, i.e., the algorithm first constructs T^1_L and subsequently constructs $\Pi^1_L, \Pi^2_L, \ldots, \Pi^n_L$. Then, we have the following three lemmas.

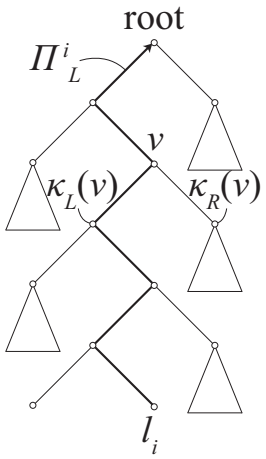

Figure 5.5: Illustration of Π^j_L

Lemma 15. *T_L and T_R can be constructed in $O(n \log n)$ time and $O(n \log n)$ space.*

Lemma 16. *For any integers i and j with $1 \leq i \leq n$ and $1 \leq j \leq n$, $\Theta^{s^i_L}_L(v_j)$ and $\Theta^{s^i_L}_R(v_j)$ can be computed in $O(\log n)$ time once T_L and T_R have been constructed.*

Lemma 17. *$\Theta^{s^i_L}_{\mathrm{opt}}$ for all i with $1 \leq i \leq n$ can be computed in $O(n \log n)$ time once T_L and T_R have been constructed.*

Proof of Lemma 15. We only prove for T_L. First, we notice that T^1_L can be constructed in $O(n)$ time and $O(n)$ space. And then, Π^1_L can be constructed in $O(\log n)$ time and $O(\log n)$ space. Suppose that $\Pi^1_L, \ldots, \Pi^{i-1}_L$ for a given i with $2 \leq i \leq n$ have been constructed. The algorithm then follows the path Π^i_L from leaf l_i to the root and stores $value(\nu; i)$ at each node ν on Π^i_L, which takes $O(\log n)$ time and $O(\log n)$ space. At leaf l_i, the algorithm sets

$$\begin{aligned} value(l_i; i) &= -v_i\tau + \sum_{1 \leq j \leq i} w^{s^i_L}(v_j) \\ &= value(l_i; 1) + \sum_{2 \leq j \leq i} (w^+(v_j) - w^-(v_j)). \end{aligned}$$

Here, suppose that the algorithm has precomputed $\sum_{2 \leq j \leq i}(w^+(v_j) - w^-(v_j))$ for all i with $2 \leq i \leq n$ in $O(n)$ time and $O(n)$ space. Suppose that for an internal node ν on Π^i_L, the algorithm has already set $value(\nu'; i)$ for every node ν' on Π^i_L between l_i and ν. Then, the algorithm sets $value(\nu; i)$ as the maximum of $value(\kappa_R(\nu); i)$ and $value(\kappa_L(\nu); i)$. If $\kappa_L(\nu)$ is on Π^i_L, since $value(\kappa_L(\nu); i)$

has already been computed, the algorithm only computes $value(\kappa_R(\nu); i)$ as

$$value(\kappa_R(\nu); i) \;=\; value(\kappa_R(\nu); 1) + \sum_{2 \le j \le i} (w^+(v_j) - w^-(v_j)).$$

If $\kappa_R(\nu)$ is on Π_L^i, since $value(\kappa_R(\nu); i)$ has already been computed, the algorithm only needs to obtain $value(\kappa_L(\nu); i)$. We notice that $i_{\max}(\kappa_L(\nu)) < i$ holds, and then, for any integer l with $1 \le l \le i_{\max}(\kappa_L(\nu))$, $w^{s_L^i}(v_l) = w^{s_L^{i_{\max}(\kappa_L(\nu))}}(v_l) = w^+(v_l)$ holds by (5.13), which implies

$$value(\kappa_L(\nu); i) = value(\kappa_L(\nu); i_{\max}(\kappa_L(\nu))). \tag{5.29}$$

This can be derived from $\Pi_L^{i_{\max}(\kappa_L(\nu))}$ (which has already been obtained). So, $\Pi_L^2, \Pi_L^3, \ldots, \Pi_L^n$ can be constructed in $O(n \log n)$ time and $O(n \log n)$ space. □

Proof of Lemma 16. We now show how to compute $\Theta_L^{s_L^i}(v_j)$ for integers i and j with $1 \le i \le n$ and $2 \le j \le n$ by using T_L (recall that we assumed $\Theta_L^s(v_1) = 0$ and $\Theta_R^s(v_n) = 0$ for any scenario s). By (5.6), we have

$$\Theta_L^{s_L^i}(v_j) = \max\{f_L^{s_L^i(k)}(v_j) \mid 1 \le k \le j-1\}. \tag{5.30}$$

Note that for any integer k with $1 \le k \le j-1$, $i_{\min}(\kappa_L(\nu)) \le k \le i_{\max}(\kappa_L(\nu))$ holds with some ν on Π_L^j. Thus, by (5.28) and (5.30), $\Theta_L^{s_L^i}(v_j)$ can be represented as

$$\Theta_L^{s_L^i}(v_j) = v_j\tau + \max\{value(\kappa_L(\nu); i) \mid \nu \text{ on } \Pi_L^j\}. \tag{5.31}$$

So, the algorithm is actually required to compute the maximum of $value(\kappa_L(\nu); i)$ for all nodes ν on Π_L^j. There are two cases: [Case 1] $j \le i$ (see Figure 5.6), and [Case 2] $j > i$ (see Figure 5.7).

[Case 1]: The algorithm follows the path Π_L^j from leaf l_j to the root. Every time the algorithm visits an internal node ν, it examines whether $\kappa_L(\nu)$ is on Π_L^j or not. If this is the case, the algorithm does nothing. Otherwise, the algorithm needs to obtain $value(\kappa_L(\nu); i)$. Recall that $i_{\max}(\kappa_L(\nu)) < i$ holds, and then, for any l with $1 \le l \le i_{\max}(\kappa_L(\nu))$, $w^{s_L^i}(v_l) = w^{s_L^{i_{\max}(\kappa_L(\nu))}}(v_l) = w^+(v_l)$ holds by (5.13), which implies

$$value(\kappa_L(\nu); i) = value(\kappa_L(\nu); i_{\max}(\kappa_L(\nu))).$$

So, the algorithm gets the rightmost leaf $l_{i_{\max}(\kappa_L(\nu))}$ in the subtree rooted at $\kappa_L(\nu)$, and retrieves $value(\kappa_L(\nu); i_{\max}(\kappa_L(\nu)))$ stored in $\Pi_L^{i_{\max}(\kappa_L(\nu))}$. The algo-

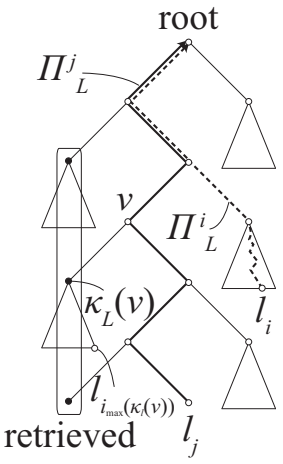

Figure 5.6: Illustration of Case 1

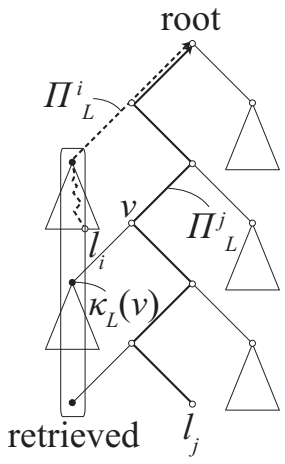

Figure 5.7: Illustration of Case 2

rithm continues to do this computation and to take the maximum value among those retrieved, which takes $O(\log n)$ time.

[Case 2]: The task the algorithm does is similar to Case 1. Every time the algorithm visits an internal node ν before Π^j_L encounters the node on Π^i_L, it examines whether $\kappa_L(\nu)$ is on Π^j_L or not. If this is the case, the algorithm does nothing. Otherwise, the algorithm retrieves $value(\kappa_L(\nu); 1)$ stored in T^1_L and adds $\sum_{2\leq j\leq i}(w^+(v_j) - w^-(v_j))$ to the retrieved value since

$$value(\kappa_L(\nu); i) = value(\kappa_L(\nu); 1) + \sum_{2\leq j\leq i}(w^+(v_j) - w^-(v_j))$$

holds. Here, suppose that the algorithm has precomputed $\sum_{2\leq j\leq i}(w^+(v_j) - w^-(v_j))$ for all i with $2 \leq i \leq n$ in $O(n)$ time. The algorithm continues to do this computation and to take the maximum value among those retrieved before encountering the node on Π^i_L, and after that, it does the same computation as in Case 1, which takes $O(\log n)$ time. □

Proof of Lemma 17. We here show how the algorithm actually computes $\Theta^{s^i_L}_{\rm opt}$ for all i with $1 \leq i \leq n$. The algorithm first computes $\Theta^{s^1_L}_{\rm opt}$ and $x^{s^1_L}_{\rm opt}$. Based on Claim 4, the algorithm can apply binary search to find adjacent two vertices v_{l-1} and v_l with some l with $2 \leq l \leq n$ such that $\Theta^{s^1_L}_L(v_{l-1}) \leq \Theta^{s^1_L}_R(v_{l-1})$ and $\Theta^{s^1_L}_L(v_l) \geq \Theta^{s^1_L}_R(v_l)$, i.e., $x^{s^1_L}_{\rm opt}$ exists in the interval $[v_{l-1}, v_l]$, which takes $O(\log^2 n)$ time by Lemma 16. We now show how to find $x^{s^1_L}_{\rm opt}$ from $[v_{l-1}, v_l]$ in constant time. Let $x(t)$ denote a point dividing $[v_{l-1}, v_l]$ with the ratio of t to $1-t$. Suppose that $0 \leq t \leq 1$. Then, by the condition of $\Theta^{s^1_L}(v_l) = \Theta^{s^1_L}_L(v_l)$

and $\Theta^{s^1_L}(v_{l-1}) = \Theta^{s^1_L}_R(v_{l-1})$, we have

$$\Theta^{s^1_L}(x(t)) = \max\{\Theta^{s^1_L}(v_{l-1}) - t(v_l - v_{l-1})\tau, \\ \Theta^{s^1_L}(v_l) - (1-t)(v_l - v_{l-1})\tau\}. \tag{5.32}$$

Let us consider the solution t^* of $\Theta^{s^1_L}(v_{l-1}) - t(v_l - v_{l-1})\tau = \Theta^{s^1_L}(v_l) - (1-t)(v_l - v_{l-1})\tau$. If $0 \leq t^* \leq 1$ holds, $x^{s^1_L}_{\rm opt} = x(t^*)$ holds by Claim 4. If $t^* < 0$ holds, then $\Theta^{s^1_L}(v_{l-1}) < \Theta^{s^1_L}(v_l) - (v_l - v_{l-1})\tau$ holds, which implies $x^{s^1_L}_{\rm opt} = v_{l-1}$. Similarly, if $t^* > 1$ holds, $x^{s^1_L}_{\rm opt} = v_l$ holds. Therefore, the algorithm can compute $\Theta^{s^1_L}_{\rm opt}$ and $x^{s^1_L}_{\rm opt}$ in $O(\log^2 n)$ time. Generally, we have the following claim.

Claim 7. *For a scenario $s \in \mathcal{S}$, suppose that $\Theta^s_L(v_{l-1}) \leq \Theta^s_R(v_{l-1})$ and $\Theta^s_L(v_l) \geq \Theta^s_R(v_l)$ hold for adjacent two vertices v_{l-1} and v_l with some l with $2 \leq l \leq n$, and let t^* denote the solution to an equation $\Theta^s(v_{l-1}) - t(v_l - v_{l-1})\tau = \Theta^s(v_l) - (1-t)(v_l - v_{l-1})\tau$. Then,*
(i) *if $0 \leq t^* \leq 1$ holds, $x^s_{\rm opt}$ is a point dividing the interval $[v_{l-1}, v_l]$ with the ratio of t^* to $1 - t^*$ and $\Theta^s_{\rm opt} = \Theta^s_R(v_{l-1}) - t^*(v_l - v_{l-1})\tau$ holds,*
(ii) *if $t^* < 0$ holds, $x^s_{\rm opt} = v_{l-1}$ and $\Theta^s_{\rm opt} = \Theta^s_R(v_{l-1})$ hold, and*
(iii) *if $t^* > 1$ holds, $x^s_{\rm opt} = v_l$ and $\Theta^s_{\rm opt} = \Theta^s_L(v_l)$ hold.*

$\Theta^{s^i_L}_{\rm opt}$ and $x^{s^i_L}_{\rm opt}$ for every i can be computed in $O(\log^2 n)$ in the same manner as $\Theta^{s^1_L}_{\rm opt}$ and $x^{s^1_L}_{\rm opt}$. Thus, $\Theta^{s^i_L}_{\rm opt}$ and $x^{s^i_L}_{\rm opt}$ for all i with $1 \leq i \leq n$ can be obtained in $O(n \log^2 n)$ time. This running time is further improved to $O(n \log n)$ time based on the nontrivial observation stated below. We first have the following claim by Lemma 14.

Claim 8. *For an integer i with $1 \leq i \leq n-1$,*
(i) *if $x^{s^i_L}_{\rm opt} \geq v_{i+1}$ holds, $v_{i+1} \leq x^{s^{i+1}_L}_{\rm opt} \leq x^{s^i_L}_{\rm opt}$ holds, and*
(ii) *if $x^{s^i_L}_{\rm opt} \leq v_{i+1}$ holds, $x^{s^i_L}_{\rm opt} \leq x^{s^{i+1}_L}_{\rm opt} \leq v_{i+1}$ holds.*

We immediately have the following claim by this claim.

Claim 9. (i) *As long as $x^{s^i_L}_{\rm opt} \geq v_{i+1}$ holds, $x^{s^i_L}_{\rm opt}$ does not increase as i increases.*
(ii) *Once $x^{s^i_L}_{\rm opt} \leq v_{i+1}$ holds, $x^{s^i_L}_{\rm opt}$ never decreases as i increases.*

For every i with $1 \leq i \leq n$, let $l(i)$ denote an integer such that $\Theta^{s^i_L}_L(v_{l(i)-1}) \leq \Theta^{s^i_L}_R(v_{l(i)-1})$ and $\Theta^{s^i_L}_L(v_{l(i)}) \geq \Theta^{s^i_L}_R(v_{l(i)})$ hold, i.e., $x^{s^i_L}_{\rm opt}$ exists in the interval $[v_{l(i)-1}, v_{l(i)}]$. Suppose that the algorithm has already computed $x^{s^i_L}_{\rm opt}$ and $[v_{l(i)-1}, v_{l(i)}]$. Then, based on Claim 8, the improved procedure which finds $[v_{l(i+1)-1}, v_{l(i+1)}]$ is described as follows.

(a) If the condition of Claim 8(i) holds, $x_{\mathrm{opt}}^{s_L^{i+1}}$ lies to the left of $v_{l(i)}$. The algorithm first tests the interval $[v_{l(i)-1}, v_{l(i)}]$ whether $\Theta_L^{s_L^{i+1}}(v_{l(i)-1}) \leq \Theta_R^{s_L^{i+1}}(v_{l(i)-1})$ and $\Theta_L^{s_L^{i+1}}(v_{l(i)}) \geq \Theta_R^{s_L^{i+1}}(v_{l(i)})$ hold or not. If this is the case, $x_{\mathrm{opt}}^{s_L^{i+1}}$ exists in $[v_{l(i)-1}, v_{l(i)}]$. Otherwise, the algorithm sequentially tests $[v_{l(i)-2}, v_{l(i)-1}], [v_{l(i)-3}, v_{l(i)-2}], \ldots$ in this order to locate $[v_{l(i+1)-1}, v_{l(i+1)}]$ where $x_{\mathrm{opt}}^{s_L^{i+1}}$ exists.
(b) If the condition of Claim 8(ii) holds, $x_{\mathrm{opt}}^{s_L^{i+1}}$ lies to the right of $v_{l(i)-1}$. The algorithm first tests the interval $[v_{l(i)-1}, v_{l(i)}]$ whether $\Theta_L(v_{l(i)-1}, s_L^{i+1}) \leq \Theta_R(v_{l(i)-1}, s_L^{i+1})$ and $\Theta_L(v_{l(i)}, s_L^{i+1}) \geq \Theta_R(v_{l(i)}, s_L^{i+1})$ hold or not. If this is the case, $x_{\mathrm{opt}}^{s_L^{i+1}}$ exists in $[v_{l(i)-1}, v_{l(i)}]$. Otherwise, the algorithm sequentially tests $[v_{l(i)}, v_{l(i)+1}], [v_{l(i)+1}, v_{l(i)+2}], \ldots$ in this order to locate $[v_{l(i+1)-1}, v_{l(i+1)}]$ where $x_{\mathrm{opt}}^{s_L^{i+1}}$ exists.

We now analyze the running time of this algorithm. Here, we recall Claim 4. For given i and j, the algorithm tests whether $x_{\mathrm{opt}}^{s_L^i}$ exists in $[v_{j-1}, v_j]$ or not by computing $\Theta_L^{s_L^i}(v_{j-1}), \Theta_R^{s_L^i}(v_{j-1}), \Theta_L^{s_L^i}(v_j)$ and $\Theta_R^{s_L^i}(v_j)$. This computation takes $O(\log n)$ time by Lemma 16. And by Claim 9, there exists an integer i' with $1 \leq i' \leq n$ such that $x_{\mathrm{opt}}^{s_L^i} \geq v_{i+1}$ holds for every $i < i'$ and $x_{\mathrm{opt}}^{s_L^{i'}} \leq v_{i'+1}$ holds. Then, we heve $l(i') \leq l(i'-1) \leq \ldots \leq l(2) \leq l(1)$ and $l(i') \leq l(i'+1) \leq \ldots \leq l(n-1) \leq l(n)$. For a given $i < i'$, the algorithm tests $l(i) - l(i+1) + 1$ intervals, i.e., $O(n)$ intervals are tested for all $i < i'$ in total. Similarly, $O(n)$ intervals are tested for all $i \geq i'$. Therefore, the overall running time is $O(n \log n)$. □

By Lemmas 15 and 17, $\Theta_{\mathrm{opt}}^{s_L^i}$ for all i with $1 \leq i \leq n$ can be computed in $O(n \log n)$ time and $O(n \log n)$ space, and $\Theta_{\mathrm{opt}}^{s_R^i}$ can be similarly treated.

Lemma 18. *$\Theta_{\mathrm{opt}}^{s_L^i}$ and $\Theta_{\mathrm{opt}}^{s_R^i}$ for all i with $1 \leq i \leq n$ can be computed in $O(n \log n)$ time and $O(n \log n)$ space.*

Phase 2

In this section, we show how to compute a minimax regret sink location x^*. Recall that binary search can be applied to find x^* by the unimodality of $R_{\max}(x)$. Actually, the algorithm finds the interval $[v_{l-1}, v_l]$ where x^* exists based on Lemma 12. For this purpose, the algorithm needs to compute a worst case scenario for v_j with any j with $1 \leq j \leq n$ as

$$\hat{s}_j = \mathrm{argmax}\{R^s(v_j) \mid s \in \mathcal{S}_L \cup \mathcal{S}_R\}$$

by evaluating $R^s(v_j)$ for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$. By the definition of (5.10), we have $R^s(v_j) = \Theta^s(v_j) - \Theta^s_{\mathrm{opt}}$. In Phase 1, Θ^s_{opt} for all $s \in \mathcal{S}_L \cup \mathcal{S}_R$ have been computed. Therefore, in Phase 2, the algorithm only needs to compute $\Theta^s(v_j)$ for each $s \in \mathcal{S}_L \cup \mathcal{S}_R$, and then, $R^s(v_j)$ can be immediately computed. Once computing $\hat{s}_j$, the algorithm can know whether $x^* \leq v_j$ or $x^* \geq v_j$ by evaluating $\Theta^{\hat{s}_j}_L(v_j)$ and $\Theta^{\hat{s}_j}_R(v_j)$ by Lemma 12.

We first show that the algorithm can compute $\Theta^s(v_j)$ for a given j with $1 \leq j \leq n$ and all $s \in \mathcal{S}_L \cup \mathcal{S}_R$ in $O(n)$ time in total. The algorithm basically needs to compute $\Theta^{s^i_L}_L(v_j)$, $\Theta^{s^i_L}_R(v_j)$, $\Theta^{s^i_R}_L(v_j)$ and $\Theta^{s^i_R}_R(v_j)$ for all i with $1 \leq i \leq n$. In the following, we only show how to compute $\Theta^{s^i_L}_L(v_j)$ for all i with $1 \leq i \leq j-1$ in $O(n)$ time in total (for i with $j \leq i \leq n$, $\Theta^{s^i_L}_L(v_j) = \Theta^{s^{j-1}_L}_L(v_j)$ holds by the definitions of (5.2) and (5.13)). Note that the other cases can be similarly treated.

For i with $1 \leq i \leq j-1$, let $id(i)$ denote the integer defined as

$$id(i) = \mathrm{argmax}\Big\{ -v_k\tau + \sum_{1 \leq l \leq k} w^{s^i_L}(v_l) \;\Big|\; 1 \leq k \leq j-1 \Big\},$$

i.e., $\Theta^{s^i_L}_L(v_j) = f^{s^i_L(id(i))}_L(v_j)$ by (5.4) and (5.6). The algorithm first computes $\Theta^{s^1_L}_L(v_j)$ and $id(1)$ by using T_L (as mentioned in Section 5.3.2), which can be done in $O(\log n)$ time by Lemma 16. Here, suppose that the algorithm has already obtained $\Theta^{s^i_L}_L(v_j)$ and $id(i)$ for a given i with $1 \leq i \leq j-1$. We then show that the algorithm can compute $\Theta^{s^{i+1}_L}_L(v_j)$ and $id(i+1)$ in constant time. There are two cases: [Case 1] $id(i) \geq i+1$, and [Case 2] $id(i) < i+1$. In the following, let $\Delta_i = w^+(v_i) - w^-(v_i)$.

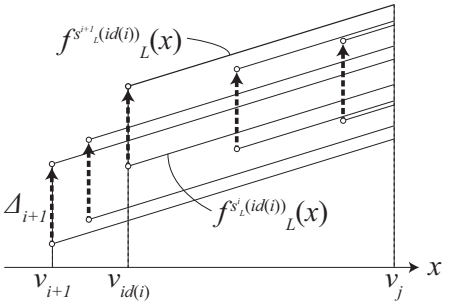

Figure 5.8: Illustration of Case 1

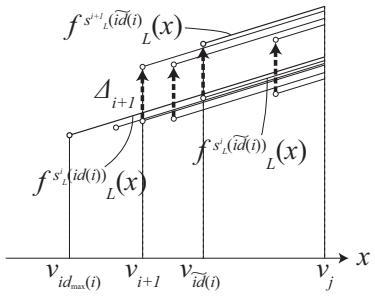

Figure 5.9: Illustration of Case 2

[Case 1]: In this case, $id(i+1)$ does not change from $id(i)$, which implies $\Theta^{s^{i+1}_L}_L(v_j) = f^{s^{i+1}_L(id(i))}_L(v_j) = f^{s^i_L(id(i))}_L(v_j) + \Delta_{i+1}$ (see Figure 5.8). We have

$f_L^{s_L^i(id(i))}(v_j) = \Theta_L^{s_L^i}(v_j)$. Thus, $\Theta_L^{s_L^{i+1}}(v_j)$ and $id(i+1)$ can be computed as

$$\begin{aligned} \Theta_L^{s_L^{i+1}}(v_j) &= \Theta_L^{s_L^i}(v_j) + \Delta_{i+1}, \text{ and} \\ id(i+1) &= id(i). \end{aligned}$$

[Case 2]: In this case, for k with $id(i) \leq k \leq i$, $f_L^{s_L^{i+1}(k)}(v_j) = f_L^{s_L^i(k)}(v_j)$ holds, and for k with $i+1 \leq k \leq j-1$, $f_L^{s_L^{i+1}(k)}(v_j) = f_L^{s_L^i(k)}(v_j) + \Delta_{i+1}$ holds (see Figure 5.9). For i with $1 \leq i \leq j-2$, we define $V(i)$ as

$$\begin{aligned} V(i) &= \max_{i+1 \leq k \leq j-1} \Big\{ -v_k\tau + \sum_{1 \leq l \leq k} w^{s_L^i}(v_l) \Big\} \\ &= \max_{i+1 \leq k \leq j-1} \Big\{ -v_k\tau + \sum_{1 \leq l \leq k} w^{s_L^1}(v_l) \Big\} + \sum_{2 \leq h \leq i} \Delta_h, \end{aligned} \tag{5.33}$$

and let $\widetilde{id}(i)$ denote the integer which attains the maximum in (5.33). When an integer j is given, $V(i)$ and $\widetilde{id}(i)$ for all i with $1 \leq i \leq j-2$ can be precomputed in $O(n)$ time as follows. The first term of (5.33) for all i with $1 \leq i \leq j-2$ can be computed in $O(n)$ time in total by computing them in descending order of i, and also, the second term of (5.33) for all i with $1 \leq i \leq j-2$ can be computed in $O(n)$ time in total by computing them in ascending order of i. Then,

$$\begin{aligned} \Theta_L^{s_L^{i+1}}(v_j) &= \max\{f_L^{s_L^{i+1}(id(i))}(v_j), f_L^{s_L^{i+1}(\widetilde{id}(i))}(v_j)\} \\ &= \max\{f_L^{s_L^i(id(i))}(v_j), f_L^{s_L^i(\widetilde{id}(i))}(v_j) + \Delta_{i+1}\} \end{aligned} \tag{5.34}$$

holds. We have $f_L^{s_L^i(id(i))}(v_j) = \Theta_L^{s_L^i}(v_j)$ and $f_L^{s_L^i(\widetilde{id}(i))}(v_j) = v_j\tau + V(i)$. By these and (5.34), once $V(i)$ and $\widetilde{id}(i)$ are computed for all i with $1 \leq i \leq j-2$ in advance, $\Theta_L^{s_L^{i+1}}(v_j)$ and $id(i+1)$ can be computed as

$$\begin{aligned} \Theta_L^{s_L^{i+1}}(v_j) &= \max\{\Theta_L^{s_L^i}(v_j), v_j\tau + V(i) + \Delta_{i+1}\}, \text{ and} \\ id(i+1) &= \begin{cases} id(i) & \text{if } \Theta_L^{s_L^i}(v_j) \geq v_j\tau + V(i) + \Delta_{i+1}, \\ \widetilde{id}(i) & \text{if } \Theta_L^{s_L^i}(v_j) < v_j\tau + V(i) + \Delta_{i+1}. \end{cases} \end{aligned}$$

In the above mentioned manner, the algorithm can compute $\Theta_L^{s_L^i}(v_j)$ for all i with $1 \leq i \leq j-1$ in $O(n)$ time. Since $\Theta_R^{s_L^i}(v_j)$, $\Theta_L^{s_R^i}(v_j)$ and $\Theta_R^{s_R^i}(v_j)$ can be similarly computed in $O(n)$ time, we have the following claim.

Claim 10. *For a given j with $1 \leq j \leq n$, let $\hat{s}_j = \text{argmax}\{R^s(v_j) \mid s \in$*

$\mathcal{S}_L \cup \mathcal{S}_R\}$. *Then,* $\hat{s}_j$ *can be computed in* $O(n)$ *time once* $\Theta^{s_L^i}_{\rm opt}$ *and* $\Theta^{s_R^i}_{\rm opt}$ *for all* i *with* $1 \leq i \leq n$ *have been computed.*

When the algorithm computes $\hat{s}_j$ for a given j with $1 \leq j \leq n$, it can know whether $x^* \leq v_j$ or $x^* \geq v_j$ by Lemma 12. Therefore, the algorithm can apply binary search to find the interval $[v_{l-1}, v_l]$ with some l with $2 \leq l \leq n$ where x^* exists such that $\Theta_L^{\hat{s}_{l-1}}(v_{l-1}) \leq \Theta_R^{\hat{s}_{l-1}}(v_{l-1})$ and $\Theta_L^{\hat{s}_l}(v_l) \geq \Theta_R^{\hat{s}_l}(v_l)$, which takes $O(n \log n)$ time by Claim 10. We now show how to find x^* from $[v_{l-1}, v_l]$ in constant time, which is similar to the manner discussed at Claim 7. Let $x(t)$ denote a point dividing the interval $[v_{l-1}, v_l]$ with the ratio of t to $1-t$. Suppose that $0 \leq t \leq 1$. Then, by the condition of $R_{\max}(v_l) = \Theta_L^{\hat{s}_l}(v_l) - \Theta^{\hat{s}_l}_{\rm opt}$ and $R_{\max}(v_{l-1}) = \Theta_R^{\hat{s}_{l-1}}(v_{l-1}) - \Theta^{\hat{s}_{l-1}}_{\rm opt}$, we have

$$\begin{aligned} R_{\max}(x(t)) = \max\{&R_{\max}(v_{l-1}) - t(v_l - v_{l-1})\tau, \\ &R_{\max}(v_l) - (1-t)(v_l - v_{l-1})\tau\}. \end{aligned} \tag{5.35}$$

Let us consider the solution t^* of $R_{\max}(v_{l-1}) - t(v_l - v_{l-1})\tau = R_{\max}(v_l) - (1-t)(v_l - v_{l-1})\tau$. If $0 \leq t^* \leq 1$ holds, $x^* = x(t^*)$ holds by Lemma 12. If $t^* < 0$ holds, then $R_{\max}(v_{l-1}) < R_{\max}(v_l) - (v_l - v_{l-1})\tau$ holds, which implies $x^* = v_{l-1}$. Similarly, if $t^* > 1$ holds, $x^* = v_l$ holds. We have the following claim.

Claim 11. *Suppose that* $\Theta_L^{\hat{s}_{l-1}}(v_{l-1}) \leq \Theta_R^{\hat{s}_{l-1}}(v_{l-1})$ *and* $\Theta_L^{\hat{s}_l}(v_l) \geq \Theta_R^{\hat{s}_l}(v_l)$ *hold for adjacent two vertices* v_{l-1} *and* v_l *with some* l *with* $2 \leq l \leq n$, *and let* t^* *denote the solution to an equation* $R_{\max}(v_{l-1}) - t(v_l - v_{l-1})\tau = R_{\max}(v_l) - (1-t)(v_l - v_{l-1})\tau$. *Then,*
(i) *if* $0 \leq t^* \leq 1$ *holds,* x^* *is a point dividing the interval* $[v_{l-1}, v_l]$ *with the ratio of* t^* *to* $1 - t^*$,
(ii) *if* $t^* < 0$ *holds,* $x^* = v_{l-1}$ *holds, and*
(iii) *if* $t^* > 1$ *holds,* $x^* = v_l$ *holds.*

Concluding the above discussion for Phase 2, we have the following lemma.

Lemma 19. *The minimax regret sink location* x^* *can be computed in* $O(n \log n)$ *time once* $\Theta^{s_L^i}_{\rm opt}$ *and* $\Theta^{s_R^i}_{\rm opt}$ *for all* i *with* $1 \leq i \leq n$ *have been computed.*

By Lemmas 18 and 19, we obtain the following theorem.

Theorem 10. *The minimax regret 1-sink location problem in a dynamic path network with uniform capacity can be solved in* $O(n \log n)$ *time and* $O(n \log n)$ *space.*

5.4 Optimal k-Sink Location Problem

In this section, we consider the optimal k-sink location problem in dynamic path networks with uniform capacity. We are given a dynamic path network with uniform capacity under fixed supplies $\mathcal{N} = (P = (V, E), w, l, c', \tau)$. Recall that for a given k-sink location $(\boldsymbol{x}, \mathcal{P})$ with a k-vertex partition $\mathcal{P} = \{V_1, \ldots, V_k\}$ and a location of k sinks $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ in a input path P, the evacuation time of $(\boldsymbol{x}, \mathcal{P})$ is represented as $\Theta(\boldsymbol{x}, \mathcal{P}) = \max\{\Theta_i(\boldsymbol{x}, \mathcal{P}) \mid 1 \leq i \leq k\}$ (refer (2.5)). We below show how structure an optimal k-sink location has. Suppose that a location of k sinks $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ is first given in P such that $v_1 \leq x_1 \leq x_2 \leq \ldots \leq x_k \leq v_n$ holds. Note that each sink can be located at any point in P. Let us consider the optimal partition for a given $\boldsymbol{x}$. Throughout this chapter, we assume that all supply units of a vertex are sent to the same sink. We call a directed path along which all supply units of a vertex are sent to a sink *evacuation path*. Then, any two evacuation paths never cross each other in an optimal evacuation (otherwise, we can realize the better or equivalent evacuation by exchanging the two destinations of crossing evacuation paths), and any evacuation path never pass through a sink. Therefore, if an k-sink location $(\boldsymbol{x}, \mathcal{P})$ is optimal, each vertex set V_i of $\mathcal{P}$ consists consecutive vertices and the rightmost vertex of V_i is adjacent to the leftmost vertex of V_{i+1} for $1 \leq i \leq k-1$. Then, we call the rightmost vertex of each vertex set *dividing vertex*. Let $\boldsymbol{d} = (d_1, d_2, \ldots, d_{k-1})$ which is a $(k-1)$-dimensional vector where d_i is the index of dividing vertex in V_i for $1 \leq i \leq k-1$. We call such a $(k-1)$-dimensional vector $(k-1)$-*divider*. In this chapter, since a $(k-1)$-divider $\boldsymbol{d}$ determines the k-partition $\mathcal{P}$, we use the notations $(\boldsymbol{x}, \boldsymbol{d})$, $\Theta_i(\boldsymbol{x}, \boldsymbol{d})$ and $\Theta(\boldsymbol{x}, \boldsymbol{d})$ to denote $(\boldsymbol{x}, \mathcal{P})$, $\Theta_i(\boldsymbol{x}, \mathcal{P})$ and $\Theta(\boldsymbol{x}, \mathcal{P})$, respectively. We notice that each sink location x_i exists in the interval $[v_{d_{i-1}+1}, v_{d_i}]$ for $1 \leq i \leq k$ where $d_0 = 0$ and $d_k = n$ if an k-sink location $(\boldsymbol{x}, \boldsymbol{d})$ is optimal.

Then, the goal is to find a k-sink location $(\boldsymbol{x}, \boldsymbol{d})$ which minimizes $\Theta(\boldsymbol{x}, \boldsymbol{d})$. Therefore, the optimal k-sink location problem in a path P is defined as follows:

$$\text{minimize } \left\{\Theta(\boldsymbol{x}, \boldsymbol{d}) \mid \boldsymbol{x} \in P^k \text{ and } \boldsymbol{d} \in \{1, 2, \ldots, n\}^{k-1}\right\}. \tag{5.36}$$

5.4.1 Recursive formulation

We now consider a subproblem of the above mentioned problem: for some integers i, j and p with $1 \leq i \leq j \leq n$ and $1 \leq p \leq k$, the p-sink location problem in $[v_i, v_j]$. For $[v_i, v_j]$, let $\boldsymbol{x}^*(p, i, j)$ denote the optimal p-sink location

and $\boldsymbol{d}^*(p,i,j)$ denote the optimal $(p-1)$-divider. Note that $\boldsymbol{x}^*(p,i,j)$ is a p-dimensional vector and $\boldsymbol{d}^*(p,i,j)$ is also a $(p-1)$-dimensional vector, so $\boldsymbol{d}^*(p,i,j)$ is not defined for $p=1$. Also, let $\mathsf{OPT}(p,i,j)$ denote the optimal cost of p-sink location in $[v_i,v_j]$, i.e., the minimum time required to send all supplies on $[v_i,v_j]$ divided by $\boldsymbol{d}^*(p,i,j)$ to $\boldsymbol{x}^*(p,i,j)$. Note that if $p \geq j-i+1$ holds, the optimal sink location is trivial, i.e., $\mathsf{OPT}(p,i,j)=0$.

Next, we show the recursive formula of $\mathsf{OPT}(p,i,j)$. For integers i,j and p with $1 \leq i \leq j \leq n$ and $1 \leq p \leq k-1$, let us consider the optimal $(p+1)$-sink location and p-divider for $[v_i,v_j]$, i.e., $\boldsymbol{x}^*(p+1,i,j)$ and $\boldsymbol{d}^*(p+1,i,j)$. Since any two evacuation paths never cross each other in an optimal evacuation, there exists an integer h with $i \leq h \leq j-1$ such that all supplies on $[v_{h+1},x_j]$ are sent to the rightmost sink and all supplies on $[x_i,v_h]$ are sent to the other k sinks. Thus, we have the following recursion:

$$\mathsf{OPT}(p+1,i,j) = \min_{i \leq h \leq j-1} \max\{\mathsf{OPT}(p,i,h), \mathsf{OPT}(1,h+1,j)\}. \tag{5.37}$$

Here, let d be an integer which minimizes the maximum of $\mathsf{OPT}(p,i,h)$ and $\mathsf{OPT}(1,h+1,j)$ on $i \leq h \leq j-1$:

$$d = \operatorname*{argmin}_{i \leq h \leq j-1} \max\{\mathsf{OPT}(p,i,h), \mathsf{OPT}(1,h+1,j)\}. \tag{5.38}$$

Then, $\boldsymbol{x}^*(p+1,i,j)$ and $\boldsymbol{d}^*(p+1,i,j)$ can be represented by using d as follows:

$$\boldsymbol{x}^*(p+1,i,j) = (\boldsymbol{x}^*(p,i,d), \boldsymbol{x}^*(1,d+1,j)), \tag{5.39}$$

$$\boldsymbol{d}^*(p+1,i,j) = (\boldsymbol{d}^*(p,i,d), d). \tag{5.40}$$

5.4.2 Properties of the 1-sink location problem in a subpath

First, we consider the evacuation time of a sink location x in an interval $[v_i,v_j]$. Let $\Theta^{i,j}(x)$ denote the minimum time required to send all supplies on $[v_i,v_j]$ to x. Here, let $\Theta^i_L(x)$ (resp. $\Theta^j_R(x)$) denote the minimum time required to send all supplies on $[v_i,x]$ (resp. $[x,v_j]$) to x where $\Theta^i_L(v_i)=0$ and $\Theta^j_R(v_j)=0$. Then, by (2.12), $\Theta^{i,j}(x)$ can be represented as follows:

$$\Theta^{i,j}(x) = \max\left\{\Theta^i_L(x), \Theta^j_R(x)\right\}. \tag{5.41}$$

Also, by (2.17) and (2.18), we have the following formulae for continuous model assuming that the uniform capacity of edge is set as $c' = 1$:

$$\Theta_L^i(x) = \max_l \left\{ (x - v_l)\tau + \sum_{i \le h \le l} w(v_h) \;\middle|\; v_l \in [v_i, x) \right\}, \tag{5.42}$$

$$\Theta_R^j(x) = \max_l \left\{ (v_l - x)\tau + \sum_{l \le h \le j} w(v_h) \;\middle|\; v_l \in (x, v_j] \right\}. \tag{5.43}$$

Note that $\Theta_L^i(x)$ (resp. $\Theta_R^j(x)$) is a piecewise linear strictly increasing (resp. decreasing) function of x. Therefore, a function $\Theta^{i,j}(x)$ is unimodal in x, and there exists the unique point which minimizes $\Theta^{i,j}(x)$, that is, $\boldsymbol{x}^*(1, i, j)$. In the same way as Claims 3 and 4, we have the following claims.

Claim 12. *For any integers i and j with $1 \le i \le j \le n$, and a point $x \in [v_i, v_j]$ a function $\Theta^{i,j}(x)$ is unimodal in x with $x \in [v_i, v_j]$.*

Claim 13. *For any integers i and j with $1 \le i \le j \le n$ and a sink location $x \in [v_i, v_j]$,*
(i) *if $\Theta_L^i(x) \ge \Theta_R^j(x)$ holds, $\boldsymbol{x}^*(1, i, j) \le x$ holds, and*
(ii) *if $\Theta_L^i(x) \le \Theta_R^j(x)$ holds, $\boldsymbol{x}^*(1, i, j) \ge x$ holds.*

In the following, when x is at a vertex v_t with $i \le t \le j$, we use the notation $L(i, t)$ (resp. $R(t, j)$) to denote the value $\Theta_L^i(v_t)$ (resp. $\Theta_R^j(v_t)$). Then, we have the following claim in the same way as Claim 7.

Claim 14. *For given integers i and j with $1 \le i \le j \le n$, suppose that for the interval $[v_l, v_{l+1}]$ with $i \le l \le j-1$, $L(i, l) \le R(l, j)$ and $L(i, l+1) \ge R(l+1, j)$ hold, and let t^* denote the solution to an equation for t: $R(l, j) - t(v_{l+1} - v_l)\tau = L(i, l+1) - (1-t)(v_{l+1} - v_l)\tau$. Then,*
(i) *if $1 \le t^* \le 1$ holds, $\boldsymbol{x}^*(1, i, j)$ is a point dividing the interval $[v_l, v_{l+1}]$ with the ratio of t^* to $1 - t^*$ and $\mathsf{OPT}(1, i, j) = R(l, j) - t^*(v_{l+1} - v_l)\tau$ holds,*
(ii) *if $t^* < 0$ holds, $\boldsymbol{x}^*(1, i, j) = v_l$ and $\mathsf{OPT}(1, i, j) = R(l, j)$ hold, and*
(iii) *if $t^* > 1$ holds, $\boldsymbol{x}^*(1, i, j) = v_{l+1}$ and $\mathsf{OPT}(1, i, j) = L(i, l+1)$ hold.*

5.4.3 Properties of the k-sink location problem

In this subsection, we show several key properties of the k-sink location problem. Here, for integers p and i with $2 \le p \le k$ and $2 \le i \le n$, let $f_{p,i}(t)$ denote

a function defined on $\{t \in \mathbb{Z} \mid 1 \leq t \leq i-1\}$:

$$f_{p,i}(t) = \max\{\mathsf{OPT}(p-1,1,t), \mathsf{OPT}(1,t+1,i)\}. \tag{5.44}$$

Note that for fixed p and i, $\mathsf{OPT}(p-1,1,t)$ is monotonically increasing in t and $\mathsf{OPT}(1,t+1,i)$ is monotonically decreasing in t. Thus, we have the following claim.

Claim 15. *For any integers p and i with $2 \leq p \leq k$ and $2 \leq i \leq n$, a function $f_{p,i}(t)$ is unimodal in t on $1 \leq t \leq i-1$.*

Let $d_{p,i}$ be an integer which minimizes $f_{p,i}(t)$ for $1 \leq t \leq i-1$:

$$d_{p,i} = \operatorname*{argmin}_{1 \leq t \leq i-1} f_{p,i}(t). \tag{5.45}$$

By Claim 15, there uniquely exists $d_{p,i}$. By (5.39) and (5.40), we have

$$\boldsymbol{x}^*(p,1,i) = (\boldsymbol{x}^*(p-1,1,d_{p,i}), \boldsymbol{x}^*(1,d_{p,i}+1,i)), \tag{5.46}$$
$$\boldsymbol{d}^*(p,1,i) = (\boldsymbol{d}^*(p-1,1,d_{p,i}), d_{p,i}). \tag{5.47}$$

Then, we prove the following two lemmas.

Lemma 20. *For any integers p and i with $2 \leq p \leq k$ and $2 \leq i \leq n-1$, $d_{p,i} \leq d_{p,i+1}$ holds.*

Lemma 21. *For any integers h, i, j and l with $1 \leq i \leq j \leq n$, $1 \leq h \leq l \leq n$, $i \leq h$ and $j \leq l$, $\boldsymbol{x}^*(1,i,j) \leq \boldsymbol{x}^*(1,h,l)$ holds.*

Proof of Lemma 20. In order to prove Lemma 20, we first confirm a fundamental property.

Claim 16. *For any integers p with $1 \leq p \leq k$, and h, i, j and l with $1 \leq h \leq i \leq j \leq l \leq n$, $\mathsf{OPT}(p,i,j) \leq \mathsf{OPT}(p,h,l)$ holds.*

We prove Lemma 20 by contradiction: there exist integers p and i with $2 \leq p \leq k$ and $2 \leq i \leq n-1$ such that $d_{p,i} > d_{p,i+1}$ holds. For ease of notation in the proof, we use the notations A, B, C, D, E and F as follows:

$$\begin{aligned} A &= \mathsf{OPT}(p-1,1,d_{p,i}), & B &= \mathsf{OPT}(1,d_{p,i}+1,i), \\ C &= \mathsf{OPT}(p-1,1,d_{p,i+1}), & D &= \mathsf{OPT}(1,d_{p,i+1}+1,i+1), \\ E &= \mathsf{OPT}(1,d_{p,i+1}+1,i), & F &= \mathsf{OPT}(1,d_{p,i}+1,i+1). \end{aligned} \tag{5.48}$$

From the assumption of $d_{p,i} > d_{p,i+1}$ and Claim 16, we can derive the following inequalities:

$$C \leq A, \tag{5.49}$$

$$B \leq E \leq D, \tag{5.50}$$

$$B \leq F \leq D. \tag{5.51}$$

Since $d_{p,i}$ minimizes $f_{p,i}(t) = \max\{\mathsf{OPT}(p-1, 1, t), \mathsf{OPT}(1, t+1, i)\}$ (refer (5.44) and (5.45)), we have the following inequality:

$$\max\{A, B\} \leq \max\{C, E\}. \tag{5.52}$$

Also, without loss of generality, we assume that $d_{p,i+1}$ is maximized unless the cost increases. By this assumption, we have the following inequality:

$$\max\{C, D\} < \max\{A, F\}. \tag{5.53}$$

Then, we consider three cases: [Case 1] $A \leq B$; [Case 2] $D \leq C$; [Case 3] $B < A$ and $C < D$.
[Case 1]: By (5.49), (5.51) and the condition of $A \leq B$, we have $C \leq A \leq F \leq D$, which contradicts (5.53).
[Case 2]: By (5.49), (5.50) and the condition of $D \leq C$, we have $B \leq E \leq C \leq A$. By this and (5.52), we have $A \leq C$. Also, by (5.49), (5.51) and the condition of $D \leq C$, we have $F \leq D \leq C \leq A$. By this and (5.53), we have $C < A$, which contradicts $A \leq C$.
[Case 3]: By (5.52) and the condition of $B < A$, we have

$$A \leq \max\{C, E\}. \tag{5.54}$$

Also, by (5.53) and the condition of $C < D$, we have

$$D < \max\{A, F\}. \tag{5.55}$$

If $F \leq A$ holds, we have $D < \max\{C, E\}$ by (5.54) and (5.55), which contradicts the condition of $C < D$ or (5.50). If $A < F$ holds, we have $D < F$ by (5.55), which contradicts (5.51). □

Proof of Lemma 21. In order to prove Lemma 21, we first confirm the following claim (refer the definitions of (5.42) and (5.43)).

Claim 17. (i) *For any integers i and j with $1 \leq j \leq i \leq n$ and any points x and y with $v_i \leq x \leq y \leq v_n$, $\Theta_L^i(x) \leq \Theta_L^j(x)$ and $\Theta_L^i(x) \leq \Theta_L^i(y)$ hold.*
(ii) *For any integers i and j with $1 \leq i \leq j \leq n$ and any points x and y with $v_1 \leq y \leq x \leq v_i$, $\Theta_R^i(x) \leq \Theta_R^j(x)$ and $\Theta_R^i(x) \leq \Theta_R^i(y)$ hold.*

We prove Lemma 21 by contradiction: there exist integers h, i, j and l with $1 \leq i \leq j \leq n$, $1 \leq h \leq l \leq n$, $i \leq h$ and $j \leq l$ such that $\boldsymbol{x}^*(1, i, j) > \boldsymbol{x}^*(1, h, l)$ holds. By this assumption, we have the following inequality:

$$i \leq h \leq \boldsymbol{x}^*(1, h, l) < \boldsymbol{x}^*(1, i, j) \leq j \leq l. \tag{5.56}$$

For ease of notation in the proof, we use the notations A, B, C, D, E, F, G and H as follows:

$$\begin{array}{ll} A = \Theta_L^i(\boldsymbol{x}^*(1, i, j)), & B = \Theta_R^j(\boldsymbol{x}^*(1, i, j)), \\ C = \Theta_L^h(\boldsymbol{x}^*(1, h, l)), & D = \Theta_R^l(\boldsymbol{x}^*(1, h, l)), \\ E = \Theta_L^i(\boldsymbol{x}^*(1, h, l)), & F = \Theta_R^j(\boldsymbol{x}^*(1, h, l)), \\ G = \Theta_L^h(\boldsymbol{x}^*(1, i, j)), & H = \Theta_R^l(\boldsymbol{x}^*(1, i, j)). \end{array} \tag{5.57}$$

From (5.56) and Claim 17, we can derive the following inequalities:

$$C \leq E \leq A, \tag{5.58}$$

$$C \leq G \leq A, \tag{5.59}$$

$$B \leq F \leq D, \tag{5.60}$$

$$B \leq H \leq D. \tag{5.61}$$

Since $\boldsymbol{x}^*(1, i, j)$ and $\boldsymbol{x}^*(1, h, l)$ are the unique points which minimize $\Theta^{i,j}(x) = \max\{\Theta_L^i(x),\ \Theta_R^j(x)\}$ and $\Theta^{h,l}(x) = \max\{\Theta_L^h(x), \Theta_R^l(x)\}$, respectively (refer (5.41)), we have the following inequalities:

$$\max\{A, B\} < \max\{E, F\}, \tag{5.62}$$

$$\max\{C, D\} < \max\{G, H\}. \tag{5.63}$$

Then, we consider three cases: [Case 1] $A \leq B$; [Case 2] $D \leq C$; [Case 3] $B < A$ and $C < D$.
[Case 1]: By (5.59), (5.61) and the condition of $A \leq B$, we have $C \leq G \leq H \leq D$, which contradicts (5.63).
[Case 2]: By (5.58), (5.60) and the condition of $D \leq C$, we have $B \leq F \leq E \leq A$, which contradicts (5.62).

[Case 3]: By (5.62) and the condition of $B < A$, we have

$$A < \max\{E, F\}. \tag{5.64}$$

Also, by (5.63) and the condition of $C < D$, we have

$$D < \max\{G, H\}. \tag{5.65}$$

If $F \leq E$ holds, we have $A < E$ by (5.64), which contradicts (5.58). Also, if $G \leq H$ holds, we have $D < H$ by (5.65), which contradicts (5.61). If $E < F$ and $H < G$ hold, we have $A < F \leq D < G$ by (5.60), (5.64) and (5.65), that is, $A < G$ holds, which contradicts (5.59). □

5.4.4 Algorithm based on dynamic programming

The algorithm basically computes $\mathsf{OPT}(1,1,1)$, ..., $\mathsf{OPT}(1,1,n)$, $\mathsf{OPT}(2,1,1)$, ..., $\mathsf{OPT}(2,1,n)$, ..., $\mathsf{OPT}(k,1,1)$, ..., $\mathsf{OPT}(k,1,n)$ in this order. For some integers p and i with $2 \leq p \leq k$ and $2 \leq i \leq n$, let us consider how to obtain $\mathsf{OPT}(p,1,i)$. Actually, in order to obtain $\mathsf{OPT}(p,1,i)$, the algorithm needs $\mathsf{OPT}(p-1,1,l)$ for $l = 1, 2, \ldots, n$ and $\mathsf{OPT}(p,1,i-1)$, which are supposed to have been obtained. By (5.37), (5.44) and (5.45), we have

$$\mathsf{OPT}(p,1,i) = f_{p,i}(d_{p,i}) = \max\{\mathsf{OPT}(p-1,1,d_{p,i}), \mathsf{OPT}(1,d_{p,i}+1,i)\}. \tag{5.66}$$

Here, we assumed that $\mathsf{OPT}(p-1,1,d_{p,i})$ has already been obtained. Thus, in order to obtain $\mathsf{OPT}(p,1,i)$, we only need to compute $\mathsf{OPT}(1,d_{p,i}+1,i)$. Recall that $d_{p,i}$ is the unique point which minimizes a function $f_{p,i}(t)$ (refer (5.44) and (5.45)). Now, the algorithm knows where $d_{p,i-1}$ exists, and by Lemma 20, $d_{p,i-1} \leq d_{p,i}$ holds. So the algorithm starts to compute $f_{p,i}(t)$ for $t = d_{p,i-1}$, and continues to compute in ascending order of t, as will be shown below. Note that a function $f_{p,i}(t)$ is unimodal in t by Claim 15, which implies that $f_{p,i}(t)$ is strictly decreasing until $t = d_{p,i}$. Thus, if the algorithm reaches the first integer $t^* \geq d_{p,i-1}$ such that $f_{p,i}(t^*) \leq f_{p,i}(t^*+1)$, it outputs t^* as $d_{p,i}$. Then, the algorithm also outputs $f_{p,i}(t^*)$ as $\mathsf{OPT}(p,1,i)$.

Computation of $f_{p,i}(t)$ for $t = d_{p,i-1}$: As above mentioned, the algorithm first computes $f_{p,i}(t)$ with $t = d_{p,i-1}$ which is defined as follows:

$$f_{p,i}(d_{p,i-1}) = \max\{\mathsf{OPT}(p-1,1,d_{p,i-1}), \mathsf{OPT}(1,d_{p,i-1}+1,i)\}. \tag{5.67}$$

Since the algorithm has already obtained $\mathsf{OPT}(p-1, 1, d_{p,i-1})$, we only need to compute $\mathsf{OPT}(1, d_{p,i-1}+1, i)$. To do this, we actually need to find $\boldsymbol{x}^*(1, d_{p,i-1}+1, i)$. On the other hand, the algorithm has already obtained $\mathsf{OPT}(p, 1, i-1)$ as follows:

$$\mathsf{OPT}(p, 1, i-1) = \max\{\mathsf{OPT}(p-1, 1, d_{p,i-1}), \mathsf{OPT}(1, d_{p,i-1}+1, i-1)\}, \quad (5.68)$$

which implies that $\boldsymbol{x}^*(1, d_{p,i-1}+1, i-1)$ has been obtained. By Lemma 21, $\boldsymbol{x}^*(1, d_{p,i-1}+1, i-1) \leq \boldsymbol{x}^*(1, d_{p,i-1}+1, i)$ holds. Let l and l' be the indices of vertices such that $\boldsymbol{x}^*(1, d_{p,i-1}+1, i-1) \in [v_l, v_{l+1}]$ with $d_{p,i-1}+1 \leq l \leq i-2$ and $\boldsymbol{x}^*(1, d_{p,i-1}+1, i) \in [v_{l'}, v_{l'+1}]$ with $d_{p,i-1}+1 \leq l' \leq i-1$, respectively (see Figure 5.10). By Claim 13, for any interval $[v_h, v_{h+1}]$ with $d_{p,i-1}+1 \leq h \leq i-1$,

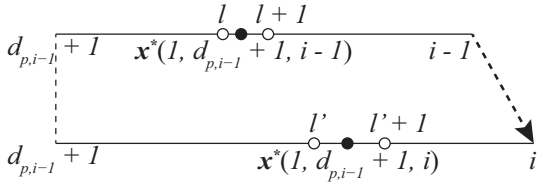

Figure 5.10: Illustrations of $\boldsymbol{x}^*(1, d_{p,i-1}+1, i-1)$ and $\boldsymbol{x}^*(1, d_{p,i-1}+1, i)$

there exists $\boldsymbol{x}^*(1, d_{p,i-1}+1, i)$ in $[v_h, v_{h+1}]$ if $L(d_{p,i-1}+1, h) \geq R(h, i)$ and $L(d_{p,i-1}+1, h+1) \leq R(h+1, i)$ hold. Therefore, if we maintain the data structure so that we can compute these values, the algorithm can test if there exists $\boldsymbol{x}^*(1, d_{p,i-1}+1, i) \in [v_h, v_{h+1}]$ or not (what the data structure is or how we can maintain and use it will be explained in the next subsection). Then, the algorithm starts to test for $h = l$, and continues to test in ascending order of h. If an interval where $\boldsymbol{x}^*(1, d_{p,i-1}+1, i)$ exists, that is, $[v_{l'}, v_{l'+1}]$ is found, then $\boldsymbol{x}^*(1, d_{p,i-1}+1, i)$ and $\mathsf{OPT}(1, d_{p,i-1}+1, i)$ can be computed in $O(1)$ time by Claim 14.

Computation of $f_{p,i}(t)$ for $t \geq d_{p,i-1}+1$: Now, suppose that for an integer t with $t \geq d_{p,i-1}$, the algorithm has already obtained $f_{p,i}(t)$, that is, $\boldsymbol{x}^*(1, t+1, i)$ and $\mathsf{OPT}(1, t+1, i)$. For an integer t with $t \geq d_{p,i-1}$, let $l(t+1)$ be the index of a vertex with $t+1 \leq l(t+1) \leq i-1$ such that $\boldsymbol{x}^*(1, t+1, i) \in [v_{l(t+1)}, v_{l(t+1)+1}]$. Note that $l(t+1)$ has also been obtained (see Figure 5.11). Then, the computation of $f_{p,i}(t+1)$ comes down to finding $l(t+2)$ which is greater than or equal to $l(t+1)$, and so, it can be treated in a similar manner to the computation of $f_{p,i}(d_{p,i-1})$.

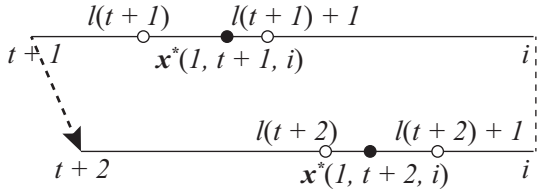

Figure 5.11: Illustrations of $\boldsymbol{x}^*(1, t+1, i)$ and $\boldsymbol{x}^*(1, t+2, i)$

5.4.5 How to compute $L(\alpha, \beta)$ and $R(\beta, \gamma)$

As mentioned in Section 5.4.4, in order to obtain $\mathsf{OPT}(p, 1, i)$ for fixed p and all $i = p+1, p+2, \ldots, n$ (note that $\mathsf{OPT}(p, 1, i) = 0$ for $i = 1, 2, \ldots, p$), the algorithm computes $f_{p,p+1}(d_{p,p}), \ldots, f_{p,p+1}(d_{p,p+1}), f_{p,p+2}(d_{p,p+1}), \ldots, f_{p,p+2}(d_{p,p+2}), \ldots, f_{p,n}(d_{p,n-1}), \ldots, f_{p,n}(d_{p,n})$. In this computation, the algorithm actually computes $L(p,p), L(p,p+1), \ldots, L(p,l(p)), L(p+1,l(p)), L(p+1,l(p)+1), \ldots, L(p+1,l(p+1)), \ldots$ where $l(i)$ is the index of vertex with $p \leq l(i) \leq l(i+1) \leq n$ for any $i \geq p$, and also, $R(p,p), R(p,p+1), \ldots, R(p,r(p)), R(p+1,r(p)), R(p+1,r(p)+1), \ldots, R(p+1,r(p+1)), \ldots$ where $r(i)$ is the index of vertex with $p \leq r(i) \leq r(i+1) \leq n$ for any $i \geq p$. In order to compute $L(\alpha, \beta)$ and $R(\beta, \gamma)$ for any integers α, β and γ with $1 \leq \alpha \leq \beta \leq \gamma \leq n$, the algorithm maintains the specific data structures $D_L(\alpha, \beta)$ and $D_R(\beta, \gamma)$, respectively. Depending on the situation, the algorithm updates $D_L(\alpha, \beta)$ to $D_L(\alpha+1, \beta)$ or $D_L(\alpha, \beta+1)$ and $D_R(\beta, \gamma)$ to $D_R(\beta+1, \gamma)$ or $D_R(\beta, \gamma+1)$. We below show definitions of the two data structures and how to maintain these.

Definition of $D_L(\alpha, \beta)$ and how to maintain $D_L(\alpha, \beta)$: In this discussion, we assume that $\alpha < \beta$ holds ($D_L(\alpha, \beta) = \emptyset$ when $\alpha = \beta$). Let us consider the evacuation of all supplies on $[v_\alpha, v_\beta]$ to v_β. We define the vertex indices $\rho_1, \ldots, \rho_e$ as

$$\begin{aligned} \rho_1 &= \operatorname{argmax}\left\{ (v_\beta - v_j)\tau + \textstyle\sum_{l=\alpha}^{j} w_l \;\middle|\; \alpha \leq j < \beta \right\} && \text{and} \\ \rho_i &= \operatorname{argmax}\left\{ (v_\beta - v_j)\tau + \textstyle\sum_{l=\rho_{i-1}+1}^{j} w_l \;\middle|\; \rho_{i-1} < j < \beta \right\} && \text{for } 2 \leq i \leq e. \end{aligned} \tag{5.69}$$

Note that $\rho_e = \beta - 1$ holds. For every integer i with $1 \leq i \leq e$, we also define the value of ρ_i as $\sigma_i = \sum\{w_h \mid \rho_{i-1} + 1 \leq h \leq \rho_i\}$ where $\rho_0 + 1 = \alpha$. Here, we notice that for every integer i with $2 \leq i \leq e$, the first unit of $v_{\rho_{i-1}}$ never be induced to stop at any vertex v_j with $\rho_{i-1} < j < \beta$. The data structure $D_L(\alpha, \beta)$ consists of the two sequences $(\rho_1, \ldots, \rho_e)$ and $(\sigma_1, \ldots, \sigma_e)$. Note that we define the size of $D_L(\alpha, \beta)$ as $|D_L(\alpha, \beta)| = e$. Recall that in continuous

model, the cost is defined on each infinitesimal unit of supply, i.e., the cost of x for a unit is defined as the minimum time required to send the unit to x. Here, we notice that for any integer i with $2 \leq i \leq e$, the first unit of $v_{\rho_{i-1}}$ never be induced to stop at v_{ρ_i}. Then, $L(\alpha, \beta)$ can be computed as

$$L(\alpha, \beta) = (v_\beta - v_{\rho_1})\tau + \sigma_1. \tag{5.70}$$

In order to update $D_L(\alpha, \beta)$ to $D_L(\alpha + 1, \beta)$, the algorithm tests if $\rho_1 = \alpha$ holds or not. If it holds, the algorithm sets $D_L(\alpha + 1, \beta)$ so that

$$\rho_i \leftarrow \rho_{i+1} \quad \text{and} \quad \sigma_i \leftarrow \sigma_{i+1} \quad \text{for} \quad 1 \leq i \leq e - 1. \tag{5.71}$$

Otherwise, the algorithm sets $D_L(\alpha + 1, \beta)$ so that

$$\begin{array}{llll} \rho_1 \leftarrow \rho_1 & \text{and} & \sigma_1 \leftarrow \sigma_1 - w_\alpha, & \\ \rho_i \leftarrow \rho_i & \text{and} & \sigma_i \leftarrow \sigma_i & \text{for} \quad 2 \leq i \leq e. \end{array} \tag{5.72}$$

On the other hand, in order to update $D_L(\alpha, \beta)$ to $D_L(\alpha, \beta + 1)$, the algorithm first sets $\rho_{e+1} = \beta$ and $\sigma_{e+1} = w_\beta$. Then, the algorithm repeatedly tests if $(v_{\rho_{j+1}} - v_{\rho_j})\tau \leq \sigma_{j+1}$ holds or not in descending order of j from $j = e$. If it holds, the algorithm sets $D_L(\alpha, \beta + 1)$ so that

$$\begin{array}{llll} \rho_i \leftarrow \rho_i & \text{and} & \sigma_i \leftarrow \sigma_i & \text{for} \quad 1 \leq i \leq j - 1, \\ \rho_j \leftarrow \rho_{j+1} & \text{and} & \sigma_j \leftarrow \sigma_j + \sigma_{j+1}, & \end{array} \tag{5.73}$$

until $(v_{\rho_{e'+1}} - v_{\rho_{e'}})\tau > \sigma_{e'+1}$ holds for $j = e'$ with some integer $e' \leq e$. Let $t(\alpha, \beta)$ denote the number of such tests required to update $D_L(\alpha, \beta)$ to $D_L(\alpha, \beta + 1)$, which can be represent as

$$t(\alpha, \beta) = e - e' + 1 = |D_L(\alpha, \beta)| - |D_L(\alpha, \beta + 1)| + 2. \tag{5.74}$$

Recall that in the computation to obtain $\mathsf{OPT}(p, 1, i)$ for fixed p and all $i = p + 1, p + 2, \ldots, n$, for each integer α with $p \leq \alpha \leq n$, the algorithm updates $D_L(\alpha, l(\alpha - 1))$ to $D_L(\alpha, l(\alpha))$ where $l(p - 1) = p$ and $l(n) = n$. Let $T(\alpha)$ denote the total number of such tests required to update $D_L(\alpha, l(\alpha - 1))$ to $D_L(\alpha, l(\alpha))$, and T denote the sum of $T(\alpha)$ for $p \leq \alpha \leq n$. By (5.71) and (5.72), we have $|D_L(\alpha, l(\alpha))| \geq |D_L(\alpha + 1, l(\alpha))|$ (assume $|D_L(\alpha + 1, \alpha)| = 0$).

By these and (5.74), the upper bound of T can be obtained as

$$\begin{aligned} T &= \sum_{\alpha=p}^{n} T(\alpha) = \sum_{\alpha=p}^{n} \sum_{\beta=l(\alpha-1)}^{l(\alpha)} t(\alpha,\beta) \\ &= \sum_{\alpha=p}^{n} \big\{ |D_L(\alpha, l(\alpha-1))| - |D_L(\alpha, l(\alpha))| + 2l(\alpha) - 2l(\alpha-1) \big\} \\ &\leq \sum_{\alpha=p}^{n} \big\{ |D_L(\alpha, l(\alpha-1))| - |D_L(\alpha+1, l(\alpha))| + 2l(\alpha) - 2l(\alpha-1) \big\} \\ &= |D_L(p, l(p-1))| + 2l(n) - 2l(p-1) = 2(n-p), \end{aligned} \tag{5.75}$$

which implies that $t(\alpha,\beta)$ is amortized $O(1)$.

Definition of $D_R(\beta,\gamma)$ and how to maintain $D_R(\beta,\gamma)$: In this discussion, we assume that $\beta < \gamma$ holds ($D_R(\beta,\gamma) = \emptyset$ when $\beta = \gamma$). Let us consider the evacuation of all supplies on $[v_\beta, v_\gamma]$ to v_β. We define the vertex indices $\mu_1, \ldots, \mu_f$ as

$$\begin{aligned} \mu_1 &= \operatorname{argmax} \left\{ (v_j - v_\beta)\tau + \textstyle\sum_{l=j}^{\gamma} w_l \;\middle|\; \beta < j \leq \gamma \right\} && \text{and} \\ \mu_i &= \operatorname{argmax} \left\{ (v_j - v_{\mu_{i-1}})\tau + \textstyle\sum_{l=j}^{\gamma} w_l \;\middle|\; \mu_{i-1} < j \leq \gamma \right\} && \text{for } 2 \leq i \leq f. \end{aligned} \tag{5.76}$$

Note that $\mu_f = \gamma$ holds. For every integer i with $1 \leq i \leq f$, we also define the value of μ_i as $W_i = \sum\{w_h \mid \mu_i \leq h \leq n\}$. Here, we notice that v_{μ_i} is the rightmost vertex of which the first unit never be induced to stop at any vertex v_j with $\mu_{i-1} < j < \mu_i$ where $\mu_0 = \beta$. In addition, let $os(\gamma) = \sum\{w_h \mid \gamma + 1 \leq h \leq n\}$. The data structure $D_R(\beta,\gamma)$ consists of the offset value $os(\gamma)$ and the two sequences $(\mu_1, \ldots, \mu_f)$ and $(W_1, \ldots, W_f)$. Note that we define the size of $D_R(\beta,\gamma)$ as $|D_R(\beta,\gamma)| = f$. Then, $R(\beta,\gamma)$ can be computed as

$$R(\beta,\gamma) = (v_{\mu_1} - v_\beta)\tau + W_1 - os(\gamma). \tag{5.77}$$

In order to update $D_R(\beta,\gamma)$ to $D_R(\beta{+}1,\gamma)$, the algorithm tests if $\mu_1 = \beta{+}1$ holds or not. If it holds, the algorithm sets $D_R(\beta+1,\gamma)$ so that

$$\mu_i \leftarrow \mu_{i+1} \quad \text{and} \quad W_i \leftarrow W_{i+1} \quad \text{for} \quad 1 \leq i \leq f-1. \tag{5.78}$$

Otherwise, nothing changes, that is, the algorithm sets $D_R(\beta{+}1,\gamma) = D_R(\beta,\gamma)$.

On the other hand, in order to update $D_R(\beta,\gamma)$ to $D_R(\beta,\gamma+1)$, the algo-

rithm first sets $\mu_{f+1} = \gamma + 1$ and compute $W_{f+1} = W_f - w_\gamma$ and $os(\gamma+1) = os(\gamma) - w_{\gamma+1}$. Then, the algorithm repeatedly tests if $\tau v_{\gamma+1} + w_{\gamma+1} \geq \tau v_{\mu_j} + W_j - os(\gamma+1)$ holds or not in descending order of j from $j = f$. If it holds, the algorithm sets $D_R(\beta, \gamma+1)$ so that

$$\begin{array}{llllll} \mu_i \leftarrow \mu_i & \text{and} & W_i \leftarrow W_i & \text{for} & 1 \leq i \leq j-1, \\ \mu_j \leftarrow \mu_{j+1} & \text{and} & W_j \leftarrow W_{j+1}, & & \end{array} \tag{5.79}$$

until $\tau v_{\gamma+1} + w_{\gamma+1} < \tau v_{\mu_{f'}} + W_{f'} - os(\gamma+1)$ holds for $j = f'$ with some integer $f' \leq f$. Let $t'(\beta, \gamma)$ denote the number of such tests required to update $D_R(\beta, \gamma)$ to $D_R(\beta, \gamma+1)$, which can be represent as

$$t'(\beta, \gamma) = f - f' + 1 = |D_R(\beta, \gamma)| - |D_R(\beta, \gamma+1)| + 2, \tag{5.80}$$

which is amortized $O(1)$ by the same discussion as that for $t(\alpha, \beta)$ defined at (5.74).

Claim 18. *For any integers α, β and γ with $1 \leq \alpha \leq \beta \leq \gamma \leq n$, $L(\alpha, \beta)$ and $R(\beta, \gamma)$ can be computed in $O(1)$ time once $D_L(\alpha, \beta)$ and $D_R(\beta, \gamma)$ have been obtained.*

Claim 19. (i) *For any integers α, β and γ with $1 \leq \alpha < \beta < r \leq n$, $L(\alpha, \beta)$ and $R(\beta, \gamma)$ can be updated to $L(\alpha+1, \beta)$ and $R(\beta+1, \gamma)$ in amortized $O(1)$ time, respectively.*
(ii) *For any integers α, β and γ with $1 \leq \alpha \leq \beta \leq \gamma \leq n-1$, $L(\alpha, \beta)$ and $R(\beta, \gamma)$ can be updated to $L(\alpha, \beta+1)$ and $R(\beta, \gamma+1)$ in amortized $O(1)$ time, respectively.*

5.4.6 Time complexity

As mentioned in Section 5.4.4 and at the beginning of Section 5.4.5, in order to obtain $\mathsf{OPT}(p, 1, i)$ for fixed p and all $i = p+1, p+2, \ldots, n$, $O(n)$ intervals are tested in total as follows: in order to test if there exists $\boldsymbol{x}^*(1, i, j)$ in an interval $[v_h, v_{h+1}]$ or not, the algorithm needs to confirm that $L(i, h) \geq R(h, j)$ and $L(i, h+1) \leq R(h+1, j)$ hold by Claim 13, which takes $O(1)$ time once $D_L(i, h), D_L(i, h+1), D_R(h, j)$ and $D_R(h+1, j)$ have been obtained by Claim 18. Thus, such computations take $O(n)$ time in total.

On the other hand, let us consider the total time required to update the data structures. For fixed p and i, when $\mathsf{OPT}(p, 1, i-1)$ is obtained, the algorithm maintains $D_L(d_{p,i-1}+1, l), D_L(d_{p,i-1}+1, l+1), D_R(l, i-1)$ and

$D_R(l+1, i-1)$, where $\boldsymbol{x}^*(1, d_{p,i-1}+1, i-1)$ exists in $[v_l, v_{l+1}]$. When $\mathsf{OPT}(p, 1, i)$ is obtained after repeatedly updating these four vertex sets, the algorithm maintains $D_L(d_{p,i}+1, l')$, $D_L(d_{p,i}+1, l'+1)$, $D_R(l', i)$ and $D_R(l'+1, i)$, where $\boldsymbol{x}^*(1, d_{p,i}+1, i)$ exists in $[v_{l'}, v_{l'+1}]$. Recall that $d_{p,i-1} \leq d_{p,i}$ and $l \leq l'$ hold by Lemmas 20 and 21. Thus, in order to obtain $\mathsf{OPT}(p, 1, i)$, the algorithm updates the four vertex sets $2(d_{p,i} - d_{p,i-1}) + 4(l' - l) + 2$ times, and so, for fixed p and all $i = 1, 2, \ldots, n$, the algorithm updates these sets $O(n)$ times in total, which takes $O(n)$ time by Claim 19.

Therefore, $\mathsf{OPT}(p, 1, i)$ for all $i = 1, 2, \ldots, n$ and $p = 1, 2, \ldots, k$ can be obtained in $O(kn)$ time.

Theorem 11. *The optimal k-sink location problem in a dynamic path network with uniform capacity can be solved in $O(kn)$ time.*

5.4.7 Extension to the case with general capacities

In this subsection, we show that the optimal k-sink location problem in a dynamic path network with general capacities can be solved in $O(kn^2 \log n)$ time.

We are given a dynamic path network with general capacities under fixed supplies $\mathcal{N} = (P = (V, E), w, l, c, \tau)$, where c is a function which associates each edge $e \in E$ with a positive capacity. Referring (5.37), the optimal k-sink location problem in a dynamic path network can be represented by the recursive formulation not only for the case with uniform capacity, but for the case with general capacities. Therefore, we can apply dynamic programming to solve the problem for the case with general capacities. If we carefully observe the proofs of Lemmas 20 and 21, we can confirm that these two lemmas still work even if edge capacities are not uniform, thus the algorithm needs to solve the optimal 1-sink location problems in subpaths $O(kn)$ times throughout the process of dynamic programming. Let us consider the optimal 1-sink location problem in an interval $[v_i, v_j]$. Suppose that a sink location x is given in $[v_i, v_j]$. We define $\Theta^{i,j}(x)$ as the minimum time required to send all supplies on $[v_i, v_j]$ to x in a similar manner to the case of uniform capacity. We also define $\Theta^i_L(x)$ (resp. $\Theta^j_R(x)$) as the minimum time required to send all supplies on $[v_i, x]$ (resp. $[x, v_j]$) to x. Then, in the same manner as (5.41), $\Theta^{i,j}(x)$ can be represented as follows:

$$\Theta^{i,j}(x) = \max\left\{\Theta^i_L(x), \Theta^j_R(x)\right\}. \tag{5.81}$$

Since $\Theta_L^i(x)$ is a monotonically increasing function in x and $\Theta_R^j(x)$ is a monotonically decreasing function in x, $\Theta^{i,j}(x)$ is an unimodal function in x, which implies that binary search can be applied to find the optimal sink location which minimizes $\Theta^{i,j}(x)$ in $[v_i, v_j]$. By the formulae (2.31) and (2.32), $\Theta^{i,j}(x)$ for a given $x \in [v_i, v_j]$ can be computed in $O(n)$ time, therefore, it takes $O(n \log n)$ time to solve the optimal 1-sink location problem in a subpath.

From the above discussion, we can derive that the overall running time is $O(kn^2 \log n)$ time to solve the optimal k-sink location problem in a dynamic path network with general capacities. We have the following theorem.

Theorem 12. *The optimal k-sink location problem in a dynamic path network with general capacities can be solved in $O(kn^2 \log n)$ time.*

5.5 Minimax Regret k-Sink Location Problem

In this section, we consider the minimax regret k-sink location problem in dynamic path networks with uniform capacity. Recently, Arumugam et al. [2] had studied this problem, and proposed the $O(kn^3 \log n)$ algorithm which uses the algorithm for the optimal k-sink location problem proposed by [D] as subroutine. Note that [D] is the preliminary version of [E] on which Section 5.4 is based. In this section, we first introduce the basic idea of [2], and then show that the result of [2] can be improved by using our new algorithm for the optimal k-sink location problem proposed in Section 5.4.

5.5.1 Properties shown by [2]

We are given a dynamic path network with uniform capacity under uncertain supplies $\mathcal{N} = (P = (V, E), W, l, c', \tau)$. Referring (2.6), let us consider a scenario $s \in \mathcal{S}$ such that for integers i and j with $1 \leq i \leq j \leq n$, $w^s(v_l) = w^+(v_l)$ for $i \leq l \leq j$ and $w^s(v_l) = w^-(v_l)$ for $1 \leq l \leq i-1$ or $j+1 \leq l \leq n$ hold. Let $\mathcal{S}_{\rm d}$ denote the set of such scenarios. The following claim was proved by [2].

Claim 20. [2] *For a k-sink location $(\boldsymbol{x}, \mathcal{P})$, there exists a worst case scenario which belongs to $\mathcal{S}_d$.*

Note that there are $O(n^2)$ scenarios in $\mathcal{S}_d$.

Next, let us consider the subproblem, that is, the minimax regret 1-sink location problem in an interval $[v_i, v_j]$. Suppose that a scenario $s \in \mathcal{S}$ and a sink location $x \in [v_i, v_j]$ are given. Referring (5.1) or (5.41), let $\Theta^{s(i,j)}(x)$

denote the minimum time required to send all supplies on $[v_i, v_j]$ under s to x. Referring (5.9), let $\Theta^{s(i,j)}_{\text{opt}}$ denote the optimal evacuation time for $[v_i, v_j]$ under s:

$$\Theta^{s(i,j)}_{\text{opt}} = \min\{\Theta^{s(i,j)}(x) \mid x \in [v_i, v_j]\}. \tag{5.82}$$

Referring (5.10), let $R^{s(i,j)}(x)$ denote the regret of x for $[v_i, v_j]$ under s:

$$R^{s(i,j)}(x) = \Theta^{s(i,j)}(x) - \Theta^{s(i,j)}_{\text{opt}}. \tag{5.83}$$

Referring (5.11), let $R^{i,j}_{\text{max}}(x)$ denote the maximum regret of x for $[v_i, v_j]$:

$$R^{i,j}_{\text{max}}(x) = \max_{s \in \mathcal{S}} R^{s(i,j)}(x). \tag{5.84}$$

Here, let $R^{i,j}_{\text{opt}}$ denote the minimax regret for $[v_i, v_j]$:

$$R^{i,j}_{\text{opt}} = \min\{R^{i,j}_{\text{max}}(x) \mid x \in [v_i, v_j]\}. \tag{5.85}$$

Then, the algorithm by [2] consists of the following three phases.

Phase 1: Compute the optimal evacuation times of k-sink location in P for all scenarios in $\mathcal{S}_d$.

Phase 2: Compute $R^{i,j}_{\text{opt}}$ for all i and j with $1 \leq i \leq j \leq n$.

Phase 3: Find the minimax regret k-sink location.

5.5.2 Improvement of the time complexity by [2]

The authors of [2] showed that the running times are $O(kn^3 \log n)$, $O(n^3)$ and $O(kn)$ for Phase 1, Phase 2 and Phase 3, respectively, that is, the overall running time is dominated by Phase 1. In Phase 1, they solve the optimal k-sink location problems $O(n^2)$ times since there are $O(n^2)$ scenarios in $\mathcal{S}_d$. For each optimal k-sink location problem, they use the $O(kn \log n)$ algorithm proposed by [D] as subroutine, so Phase 1 takes $O(kn^3 \log n)$. If we use the $O(kn)$ algorithm proposed in Section 5.4 instead of the one by [D], the overall running time can be reduced to $O(kn^3)$ time. We have the following theorem.

Theorem 13. *The minimax regret k-sink location problem in a dynamic path network with uniform capacity can be solved in $O(kn^3)$ time.*

5.6 Conclusion

In Section 5.3, we proposed an $O(n \log n)$ time algorithm for the minimax regret 1-sink location problem in dynamic path networks with uniform capacity. In Section 5.4, we proposed an $O(kn)$ time algorithm for the optimal k-sink location problem in dynamic path networks with uniform capacity. Also, in Section 5.4, we proved that the optimal k-sink location problem in a dynamic path network with general capacities can be solved in $O(kn^2 \log n)$ time. In Section 5.5, we proved that the minimax regret k-sink location problem in dynamic path networks with uniform capacity can be solved in $O(kn^3)$ time by using the algorithm proposed in Section 5.4 as a subroutine.

On the other hand, the minimax regret sink location problem in dynamic path networks with general capacities is still open. For the case with general capacities, no one has found any property reducing the number of scenarios which consists of at least one worst case scenario to polynomial size (as Lemma 13) although we have proved the formulae for the evacuation time.

Chapter 6

Minimax Regret Sink Location Problem in Dynamic Tree Networks

6.1 Introduction

In this chapter, we consider the minimax regret 1-sink location problem in dynamic tree networks with uniform capacity. Here, we assume the continuous model. Such tree structures often appears in modeling building corridors and city streets. In this chapter, we will develop a polynomial algorithms for the problem.

6.2 Outline

In Section 6.3, we consider the optimal 1-sink location problem in dynamic tree networks with uniform capacity and propose an algorithm for the problem in order to develop an algorithm for the minimax regret 1-sink location problem. In Section 6.4, we consider the minimax regret 1-sink location problem in dynamic tree networks with uniform capacity and propose an algorithm for the problem, which uses the algorithm proposed in Section 6.3 as a subroutine.

6.3 Optimal 1-Sink Location Problem

In this section, we consider the optimal 1-sink location problem in dynamic tree networks with uniform capacity. We are given a dynamic tree network with

uniform capacity under fixed supplies $\mathcal{N}_f = (T = (V, E), w, l, c', \tau)$. Referring (2.5), the evacuation time of a sink location $x \in T$ is represented as $\Theta(\boldsymbol{x}, \mathcal{P})$ where $\boldsymbol{x} = \{x\}$ and $\mathcal{P} = \{V\}$. For simplicity, in this section, we use the notation $\Theta(x)$ to denote the evacuation time of a sink location x. By (2.33) and (2.36), we can give the formula of $\Theta(x)$.

6.3.1 Properties

In this section, we prove the two lemmas which are key to our algorithm. Let x_{opt} denote a point in T which minimizes $\Theta(x)$. For two vertices $v, v' \in V$, let $V(v, v')$ denote the set of all vertices in $T(v, v')$ and $T(V')$ denote a subgraph induced by a vertex set $V' \subseteq V$.

Lemma 22. *Along a path from a leaf to another leaf in T, function $\Theta(x)$ is unimodal in x.*

Lemma 23. *For a vertex $v \in V$, if $\hat{u} = \mathrm{argmax}\{\Theta(v, u) \mid u \in \delta(v)\}$ holds, there exists $x_{\mathrm{opt}} \in T(V(v, \hat{u}) \cup \{v\})$.*

In the proofs of these two lemmas, we use the following notations. Let P be a simple path in T from a leaf to another leaf, which is represented as the sequence of vertices $v_1, v_2, \ldots, v_k$ where v_1 and v_k are leaves. In the following, for a point $p \in P$, we abuse the notation p to denote $d(v_1, p)$. For a point $p \in P$, we call the direction to v_1 (resp. v_k) from p the *left direction* (resp. *right direction*). If a sink location x is given at a vertex v_i for some i with $2 \leq i \leq k$ (resp. $1 \leq i \leq k-1$), let $\Theta_L(x; P)$ (resp. $\Theta_R(x; P)$) denote the minimum time required to complete the evacuation to x for all evacuees on $T(x, v_{i-1})$ (resp. $T(x, v_{i+1})$). If x is given on an edge $v_i v_{i+1}$ for some i with $1 \leq i \leq k-1$, let $\Theta_L(x; P)$ (resp. $\Theta_R(x; P)$) denote the minimum time required to complete the evacuation to x for all evacuees on $T(x, v_i)$ (resp. $T(x, v_{i+1})$). Also, for a vertex v_i with $1 \leq i \leq k-1$, let

$$\Theta_L^{+0}(v_i; P) = \lim_{\epsilon \to +0} \big\{\Theta_L(v_i + \epsilon; P)\big\}, \tag{6.1}$$

$$\Theta_R^{-0}(v_i; P) = \lim_{\epsilon \to +0} \big\{\Theta_R(v_i - \epsilon; P)\big\}. \tag{6.2}$$

We first show the following claim.

Claim 21. *Along a path P, function $\Theta_L(x; P)$ is increasing in x and function $\Theta_R(x; P)$ is decreasing in x.*

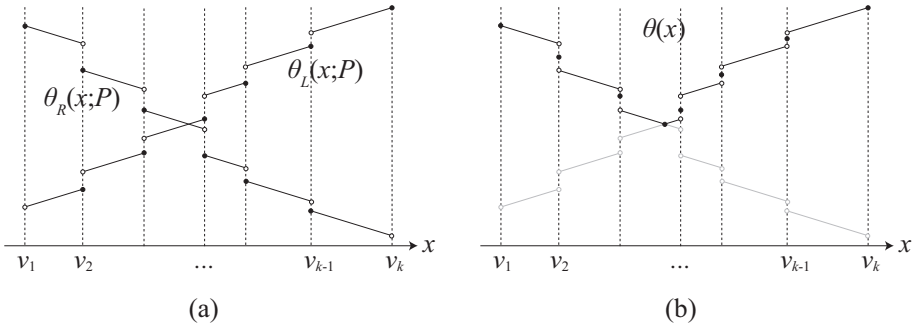

Figure 6.1: Functions along P: **(a)** $\Theta_L(x;P)$, $\Theta_R(x;P)$ and **(b)** $\Theta(x)$

Proof. By (2.36), (6.1) and (6.2), we can see the following three properties of $\Theta_L(x;P)$ and $\Theta_R(x;P)$ (see Fig. 6.1(a)): (i) for an open interval (v_{i-1}, v_i) with $2 \leq i \leq k$, $\Theta_L(x;P)$ (resp. $\Theta_R(x;P)$) is linear in x with slope τ (resp. $-\tau$), (ii) $\Theta_L(x;P)$ (resp. $\Theta_R(x;P)$) is left-continuous (resp. right-continuous) at $x = v_i$ for $2 \leq i \leq k$ (resp. $1 \leq i \leq k-1$), (iii) $\Theta_L(v_i;P) \leq \Theta_L^{+0}(v_i;P)$ (resp. $\Theta_R^{-0}(v_i;P) \geq \Theta_R(v_i;P)$) holds at v_i for $2 \leq i \leq k-1$. From these properties, $\Theta_L(x;P)$ (resp. $\Theta_R(x;P)$) is piecewise linear increasing (resp. decreasing) in x. □

By Claim 21, there uniquely exists $x \in P$ which minimizes $\max\{\Theta_L(x;P), \Theta_R(x;P)\}$, called $x_{\rm opt}(P)$ in the following. Then, we have the following claim.

Claim 22. (i) *For a vertex $v_i \in P$ such that $v_i \geq x_{\rm opt}(P)$, $\Theta_L(v_i;P) \leq \Theta(v_i) \leq \Theta_L^{+0}(v_i;P)$.* (ii) *For a vertex $v_i \in P$ such that $v_i \leq x_{\rm opt}(P)$, $\Theta_R^{-0}(v_i;P) \geq \Theta(v_i) \geq \Theta_R(v_i;P)$.*

Proof. Here, we prove only (i) ((ii) can be similarly proved). Let us look at a vertex $v_i \in P$ such that $v_i \geq x_{\rm opt}(P)$ (see Fig. 6.1(b)). By definition of $\Theta(v_i)$, we have $\Theta(v_i) \geq \Theta_L(v_i;P)$. Thus, in order to prove (i), we only need to show that

$$\Theta(v_i) \leq \Theta_L^{+0}(v_i;P). \tag{6.3}$$

By the condition of $v_i \geq x_{\rm opt}(P)$, $\Theta_L^{+0}(v_i;P) \geq \Theta_R(v_i;P)$ holds. Therefore, if $\Theta(v_i) = \Theta_R(v_i;P)$, (6.3) holds. If $\Theta(v_i) = \Theta_L(v_i;P)$, (6.3) also holds by (2.36) and (6.1). Otherwise, for a sink location $x = v_i$, an evacuee who lastly reaches v_i arrives at v_i through some adjacent vertex $u \in \delta(v_i)$ which is not on P. Suppose that we move the sink location from $x = v_i$ towards a point along P with distance ϵ in the right direction (i.e., $x = v_i + \epsilon$) where ϵ is a sufficiently small positive number. Then, the last evacuee first reaches v_i at time $\Theta(v_i)$,

may be blocked there, and eventually reaches $x = v_i + \epsilon$, thus, he/she can reach $x = v_i + \epsilon$ after time $\Theta(v_i) + \epsilon\tau$, that is, $\Theta(v_i) + \epsilon\tau \leq \Theta_L(v_i + \epsilon; P)$ holds. By definition of (6.1), we obtain (6.3). □

Proof of Lemma 22. By Claims 21 and 22, we obtain that $\Theta(x)$ may possibly be discontinuous at v_i for $2 \leq i \leq k-1$ but it is always unimodal in x along P. □

Proof of Lemma 23. Let us consider a path P from a leaf to another leaf through adjacent vertices v and $\hat{u}$ where $\hat{u} = \text{argmax}\{\Theta(v, u) \mid u \in \delta(v)\}$. Let us define the left direction in P as the direction from v to $\hat{u}$ and the right direction as the other one. Suppose that there are $k+1$ vertices $v_1, v_2, \ldots, v_k$ in P, and $v = v_i$ and $\hat{u} = v_{i-1}$ for some i with $2 \leq i \leq k-1$. We consider a point $p \in P$ such that $p = v_i + \epsilon$ with sufficiently small $\epsilon > 0$. If we can show $\Theta(v_i) < \Theta(p)$, there never exists x_{opt} in the right direction from v_i along P by Lemma 22. Then, this lemma can be proved by repeatedly applying the same discussion to all the other paths through v and $\hat{u}$. By the assumption of $\Theta(v_i) = \Theta_L(v_i; P)$, $\Theta_L(v_i; P) \geq \Theta_R(v_i; P)$ holds, and by (2.36), $\Theta_L(v_i; P) + \epsilon\tau \leq \Theta_L(p; P)$ and $\Theta_R(v_i; P) = \Theta_R(p; P) + \epsilon\tau$, that is, $\Theta_L(v_i; P) < \Theta_L(p; P)$ and $\Theta_R(v_i; P) > \Theta_R(p; P)$ hold. Thus, we have $\Theta_R(p; P) < \Theta_L(p; P)$, which implies that

$$\Theta(p) = \Theta_L(p; P). \tag{6.4}$$

From (6.4) and the above mentioned two facts $\Theta(v_i) = \Theta_L(v_i; P)$ and $\Theta_L(v_i; P) < \Theta_L(p; P)$, we derive $\Theta(v_i) < \Theta(p)$. □

6.3.2 Algorithm

In this section, we present an $O(n \log n)$ time algorithm for the optimal sink location problem in dynamic tree networks with uniform capacity, which we call BST (Binary Search in Tree).

First, we introduce the concept of *centroid of a tree* [31].

Definition 1. *For an undirected tree $T = (V, E)$, a centroid of T is a vertex which minimizes* $\max\{|V(v, u)| \mid u \in \delta(v)\}$ *for all $v \in V$.*

Kang et al. [31] showed that a centroid m of T can be computed in $O(|V|)$ time and

$$\max\{|V(m, u)| \mid u \in \delta(m)\} \leq \frac{|V|}{2} \tag{6.5}$$

holds.

Let us explain at the first iteration by algorithm BST. Letting $U_1 = V$, the algorithm first finds a centroid m_1 of $T(U_1)$ and computes $d(m_1, v)$ for every $v \in U_1$. Then, in order to compute $\Theta(m_1, u)$ for each $u \in \delta(m_1)$, the algorithm basically creates the list $L(u)$ of all vertices $v \in U_1 \cap V(m_1, u)$ which are arranged in the nondecreasing order of $d(m_1, v)$. From (2.36), we can derive that $\Theta(m_1, u)$ can be computed by using $L(u)$. In this manner, the algorithm computes $u_1 = \mathrm{argmax}\{\Theta(m_1, u) \mid u \in \delta(m_1)\}$. After that, it sets $V_1 = U_1 \setminus (V(m_1, u_1) \cup \{m_1\})$ and merges lists $L(u)$ for $u \in \delta(m_1) \setminus \{u_1\}$ into a new list L_1. At the end of the first iteration, the algorithm sets $U_2 = U_1 \cap (V(m_1, u_1) \cup \{m_1\})$. Note that by Lemma 23, there exists x_{opt} in $T(U_2)$ and by (6.5), $|U_2| \leq |U_1|/2 + 1$ holds.

The algorithm iteratively performs the same procedure (see Fig. 6.2). Namely,

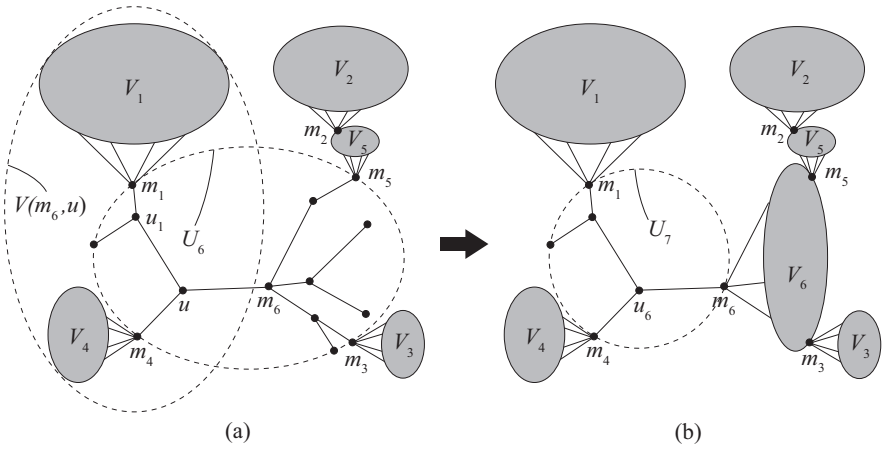

Figure 6.2: Illustration of the i-th iteration: **(a)** $i = 6$ and **(b)** $i = 7$

at the i-th iteration, it finds a centroid m_i of $T(U_i)$, computes $u_i = \mathrm{argmax}\{\Theta(m_i, u) \mid u \in \delta(m_i)\}$, sets $V_i = U_i \setminus (V(m_i, u_i) \cup \{m_i\})$, creates a list L_i of vertices $v \in V_i$ arranged in the nondecreasing order of $d(m_i, v)$ and also sets $U_{i+1} = U_i \cap (V(m_i, u_i) \cup \{m_i\})$. Since, at each iteration, the algorithm reduces the subgraph where x_{opt} exists so that the size becomes half or less roughly, it halts after $l = O(\log |V|)$ iterations. At this point, it finds two vertices m_l and $u_l \in U_l$ connected by an edge on which x_{opt} lies. Then, x_{opt} can be computed in a similar manner discussed at Claim 7. We have the following claim.

Claim 23. *Suppose that* $u_l = \mathrm{argmax}\{\Theta(m_l, v) \mid v \in \delta(m_l)\}$ *and* $m_l = \mathrm{argmax}\{\Theta(u_l, v) \mid v \in \delta(u_l)\}$ *hold for adjacent two vertices* m_l *and* u_l*, and let* t^* *denote the solution to an equation* $\Theta(m_l, u_l) - td(m_l, u_l)\tau = \Theta(u_l, m_l) - (1-t)d(m_l, u_l)\tau$*. Then,*

(i) *if* $0 \leq t^* \leq 1$ *holds,* $x_{\rm opt}$ *is a point dividing the edge* $m_l u_l$ *with the ratio of* t^* *to* $1 - t^*$ *and* $\Theta_{\rm opt} = \Theta(m_l, u_l) - t^* d(m_l, u_l)\tau$ *holds,*
(ii) *if* $t^* < 0$ *holds,* $x_{\rm opt} = m_l$ *and* $\Theta_{\rm opt} = \Theta(m_l)$ *hold, and*
(iii) *if* $t^* > 1$ *holds,* $x_{\rm opt} = u_l$ *and* $\Theta_{\rm opt} = \Theta(u_l)$ *hold.*

Now, let us analyze the time complexity of algorithm BST. We first show that the running time is $O(n \log^2 n)$ which will be improved to $O(n \log n)$ later, where $n = |V|$. Let us examine the running time for each iteration required by the algorithm. At the i-th iteration for $i \geq 2$, a centroid m_i of $T(U_i)$ can be found in $O(|U_i|)$ time (in [31]), and $d(m_i, v)$ can be computed for all $v \in V$ by depth-first search in $O(n)$ time. In the following, we consider two lists of vertices in $V(m_i, u)$ for $u \in \delta(m_i)$ which are arranged in the nondecreasing order of the distance from m_i, that is, $L(u)$ and $L'(u)$. Only one difference between $L(u)$ and $L'(u)$ is that $L(u)$ just consists of vertices in $U_i \cap V(m_i, u)$ although $L'(u)$ consists of all vertices in $V(m_i, u)$. If the algorithm creates a list $L'(u)$, $\Theta(m_i, u)$ can be computed as mentioned above. Each list $L'(u)$ can be created by a simple merge sort in $O(|V(m_i, u)| \log |V(m_i, u)|)$ time, so $u_i = \mathrm{argmax}\{\Theta(m_i, u) \mid u \in \delta(m_i)\}$ can be computed in $O(n \log n + n)$ time. Therefore, in each iteration, it takes $O(|U_i| + n + n \log n + n) = O(n \log n)$ time. Since the algorithm halts after $O(\log n)$ iterations as mentioned above, our problem can be solved in $O(n \log^2 n)$ time.

Next, we show that the running time required to create lists $L'(u)$ for $u \in \delta(m_i)$ can be improved from $O(n \log n)$ to $O(n + |U_i| \log |U_i|)$. We first show the following claim.

Claim 24. $|U_i| = O(\frac{n}{2^{i-1}})$ *and* $|V_i| = O(\frac{n}{2^{i-1}})$ *hold for* $i \geq 1$.

Proof. By definition of U_i, we can clearly see that $|U_i| = O(n/2^{i-1})$ holds. Remind that $V_i = U_i \setminus (V(m_i, u_i) \cup \{m_i\})$ and $|U_i \cap V(m_i, u_i)| = O(|U_i|/2)$, thus we have $|V_i| = O(n/2^{i-1})$. □

The idea to improve the running time is to use the sorted lists L_j with $j = 1, 2, \ldots, i-1$. Let us look at Fig. 6.2(a), and focus on a vertex $u \in \delta(m_6)$ in the figure. The computation of $L'(u)$ can be done in $O(n \log n)$ time if we know $d(m_6, v)$ for all $v \in V(m_6, u)$. But, since $V(m_6, u) = V_1 \cup V_4 \cup (U_6 \cap V(m_6, u))$ holds and we have already computed L_1 and L_4, $L'(u)$ can be obtained faster if we only create a list $L(u)$ by computing $d(m_6, v)$ for all $v \in U_6 \cap V(m_6, u)$. Note that by (6.5), $|U_6 \cap V(m_6, u)|$ is at most $|U_6|/2$, which is about $|V_1|/64$ or $|V_4|/8$ by Claim 24, so its size is much smaller than $|V(m_6, u)|$. The idea is formalized as follows. For each $u \in \delta(m_i)$, the algorithm first creates a list $L(u)$ of vertices

in $U_i \cap V(m_i, u)$, which takes $O(n' \log n')$ time where $n' = |U_i \cap V(m_i, u)|$. Thus, lists $L(u)$ for all $u \in \delta(m_i)$ can be created in $O(|U_i| \log |U_i|)$ time. For each $u \in \delta(m_i)$, the algorithm merges $L(u)$ and all lists L_j with $V_j \subseteq V(m_i, u)$ into $L'(u)$ (at this point, all of the original lists are maintained since these will be used later). For this merging operation, if we apply a simple merge sort, it takes $O(|V(m_i, u)| \log |V(m_i, u)|)$ time, which does not improve the running time. Here, we notice that $|L_j| = |V_j|$ for $1 \leq j \leq i-1$. Instead, the algorithm basically takes the following two steps to create each list $L'(u)$ for $u \in \delta(m_i)$.
[Step 1]: For L_j such that $V_j \subseteq V(m_i, u)$, choose $L_p = \text{argmin}\{|L_j| \mid V_j \subseteq V(m_i, u)\}$ and merge each L_j in the increasing order of size (i.e., the decreasing order of j) with L_p one by one.
[Step 2]: Merge the list obtained at Step 1 and $L(u)$ into $L'(u)$.
For all $u \in \delta(m_i)$, Step 1 takes in $O(\sum_{j=1}^{i-1} jn/2^{j-1}) = O(n)$ time, and thus, Step 2 takes $O(n + |U_i|) = O(n)$ time. Recall that $L(u)$ for all $u \in \delta(m_i)$ can be created in $O(|U_i| \log |U_i|)$ time. Then, by Claim 24, it takes $O(n + |U_i| \log |U_i|) = O(n + (n/2^{i-1}) \log(n/2^{i-1}))$ time to create lists $L'(u)$ for all $u \in \delta(m_i)$.

Lemma 24. *The i-th iteration of algorithm* BST *takes $O(n + \frac{n}{2^{i-1}} \log \frac{n}{2^{i-1}})$ time.*

Recall that the algorithm halts after $O(\log n)$ iterations. Thus, by Lemma 24, it takes $O(n \log n + \sum\{(n/2^{i-1}) \log(n/2^{i-1}) \mid 1 \leq i \leq \log n\}) = O(n \log n)$ time for the entire iterations. Therefore, we obtain the following theorem.

Theorem 14. *The optimal sink location problem in a dynamic tree network with uniform capacity can be solved in $O(n \log n)$ time.*

6.4 Minimax Regret 1-Sink Location Problem

In this section, we consider the minimax regret 1-sink location problem in dynamic tree networks with uniform capacity. We are given a dynamic tree network with uniform capacity under uncertain supplies $\mathcal{N}_u = (T = (V, E), W, l, c', \tau)$. Let $\mathcal{S}$ denote a set of scenarios (refer (2.6)). Referring (2.7), the evacuation time of a sink location $x \in P$ under a scenario $s \in \mathcal{S}$ is represented as $\Theta^s(\boldsymbol{x}, \mathcal{P})$ where $\boldsymbol{x} = \{x\}$ and $\mathcal{P} = \{V\}$. Suppose that a sink location $x \in T$ and a scenario $s \in \mathcal{S}$ are given. For simplicity, in this section, we use the notation $\Theta^s(x)$ to denote the evacuation time of a sink location x under a scenario s. For a

vertex $u \in \delta(x)$, let $\Theta^s(x,u)$ denote the minimum time required to send all supplies on $T(x,u)$ under s to x. Then, by (2.33), we have

$$\Theta^s(x) \quad = \quad \max\{\Theta^s(x,u) \mid u \in \delta(x)\}. \tag{6.6}$$

For $\hat{u} = \text{argmax}\{\Theta^s(x,u) \mid u \in \delta(x)\}$, we also have by (2.36)

$$\Theta^s(x,\hat{u}) = \max_{1\le i\le n'} \left\{ d(x,v_i)\tau + \sum_{i\le j\le n'} w^s(v_j) \right\}. \tag{6.7}$$

Here, let x^s_{opt} denote a point in T which minimizes $\Theta^s(x)$ under a scenario $s \in \mathcal{S}$. For a given scenario $s \in \mathcal{S}$, the optimal evacuation time Θ^s_{opt} is represented as follows (refer (2.8)):

$$\Theta^s_{\text{opt}} = \min\{\Theta^s(x) \mid x \in T\}. \tag{6.8}$$

Referring (2.9), the regret of a sink location $x \in T$ under a scenario $s \in \mathcal{S}$ is represented as $R^s(\boldsymbol{x}, \mathcal{P})$ where $\boldsymbol{x} = \{x\}$ and $\mathcal{P} = \{V\}$. For simplicity, in this section, we use the notation $R^s(x)$ to denote the regret of a sink location x under a scenario s. Then, for given a scenario $s \in \mathcal{S}$ and a sink location $x \in T$, the regret $R^s(x)$ is represented as follows:

$$R^s(x) = \Theta^s(x) - \Theta^s_{\text{opt}}. \tag{6.9}$$

We also use the notation $R_{\max}(x)$ to denote the maximum regret of a sink location x instead of $R_{\max}(\boldsymbol{x}, \mathcal{P})$ (refer (2.10)). Then, for a given sink location $x \in T$, the maximum regret $R_{\max}(x)$ is represented as follows:

$$R_{\max}(x) = \max_{s\in\mathcal{S}} R^s(x). \tag{6.10}$$

The goal is to find the minimax regret sink location x^* which minimizes $R_{\max}(x)$. Therefore, the minimax regret 1-sink location problem in a tree T is defined as follows:

$$\text{minimize } \{R_{\max}(x) \mid x \in T\}. \tag{6.11}$$

6.4.1 Properties

First, we define a set of so-called *dominant scenarios* for a vertex $v \in V$ among which the worst case scenario exists when the sink is located at v (refer 2.11).

Suppose that u is a vertex adjacent to v, n' is the number of vertices in $T(v,u)$ and $v_1(=u), v_2, \ldots, v_{n'}$ are vertices in $T(v,u)$ such that $d(v,v_i) \leq d(v,v_{i+1})$ for $1 \leq i \leq n'-1$. We now consider a scenario $s \in \mathcal{S}$ such that $w^s(v_i) = w^+(v_i)$ for $v_i \in T(v,u)$ such that $l \leq i \leq n'$ for some l with $1 \leq l \leq n'$ and $w^s(v') = w^-(v')$ for all the other vertices $v' \in V$. In the following, such a scenario is said to be *dominant* for v, and represented by $s(v,v_l)$. Then, let $\mathcal{S}_{\rm d}(v,u) = \{s(v,v_l) \mid 1 \leq l \leq n'\}$, and also let $\mathcal{S}_{\rm d}(v) = \bigcup_{u \in \delta(v)} \mathcal{S}_{\rm d}(v,u)$. Note that $\mathcal{S}_{\rm d}(v)$ consists of $n-1$ scenarios. The following is a key lemma, which can be obtained from Lemma 13.

Lemma 25. *If a sink is located at a vertex $v \in V$, there exists a worst case scenario for v which belongs to $\mathcal{S}_{\rm d}(v)$.*

Proof. Suppose that $\hat{s} = \mathrm{argmax}\{R^s(v) \mid s \in \mathcal{S}\}$, $\hat{u} = \mathrm{argmax}\{\Theta^{\hat{s}}(v,u) \mid u \in \delta(v)\}$, n' is the number of vertices in $T(v,\hat{u})$, $v_1(=\hat{u}), v_2, \ldots, v_{n'}$ are vertices in $T(v,\hat{u})$ such that $d(v,v_i) \leq d(v,v_{i+1})$ for $1 \leq i \leq n'-1$ and

$$l = \mathop{\mathrm{argmax}}_{1 \leq j \leq n'} \left\{ d(v,v_j)\tau + \sum_{j \leq i \leq n'} w^{\hat{s}}(v_i) \right\}, \tag{6.12}$$

that is,

$$\Theta^{\hat{s}}(v,\hat{u}) = d(v,v_l)\tau + \sum_{l \leq i \leq n'} w^{\hat{s}}(v_i). \tag{6.13}$$

Here, let us consider a dominant scenario $s(v,v_l)$. Then, we prove that $R^{s(v,v_l)}(v) \geq R^{\hat{s}}(v)$ holds. If $\hat{s}$ is not equal to $s(v,v_l)$, we have two cases, i.e.,

- **(I)** there exists a vertex $v' \in V(v,\hat{u})$ such that $d(v,v_l) \leq d(v,v') \leq d(v,v_{n'})$ and $w^{\hat{s}}(v') < w^+(v')$,
- **(II)** there exists a vertex $v' \in V(v,\hat{u})$ such that $d(v,v_1) \leq d(v,v') < d(v,v_l)$ and $w^{\hat{s}}(v') > w^-(v')$ or $v' \in V \setminus V(v,\hat{u})$ such that $w^{\hat{s}}(v') > w^-(v')$.

For (I), we consider another scenario $\hat{s}_+$ such that $w^{\hat{s}_+}(v') = w^+(v')$ and $w^{\hat{s}_+}(v) = w^{\hat{s}}(v)$ for $v \in V \setminus \{v'\}$. For (II), we similarly consider $\hat{s}_-$ such that $w^{\hat{s}_-}(v') = w^-(v')$ and $w^{\hat{s}_-}(v) = w^{\hat{s}}(v)$ for $v \in V \setminus \{v'\}$. If we can show that $R^{\hat{s}_+}(v) \geq R^{\hat{s}}(v)$ holds for (I) and $R^{\hat{s}_-}(v) \geq R^{\hat{s}}(v)$ holds for (II), we will eventually obtain $R^{s(v,v_l)}(v) \geq R^{\hat{s}}(v)$ by repeatedly applying the same discussion as long as there exists such a vertex v'.

(I): Let $\Delta = w^+(v') - w^{\hat{s}}(v')$. We first notice $\Theta^{\hat{s}_+}(v) = \Theta^{\hat{s}_+}(v,\hat{u})$ and $\Theta^{\hat{s}_+}(v,\hat{u}) = d(v,v_l)\tau + \sum_{l \leq i \leq n'} w^{\hat{s}_+}(v_i) = \Theta^{\hat{s}}(v,\hat{u}) + \Delta$ by (6.7) and (6.13).

Thus, we have

$$\Theta^{\hat{s}_+}(v) = \Theta^{\hat{s}}(v) + \Delta. \tag{6.14}$$

By the optimality of $x^{\hat{s}_+}_{\mathrm{opt}}$ under $\hat{s}_+$, $\Theta^{\hat{s}_+}_{\mathrm{opt}} \leq \Theta^{\hat{s}_+}(x^{\hat{s}}_{\mathrm{opt}})$ holds. Here, we claim that $\Theta^{\hat{s}_+}(p) \leq \Theta^{\hat{s}}(p) + \Delta$ holds for any point $p \in T$, so $\Theta^{\hat{s}_+}(x^{\hat{s}}_{\mathrm{opt}}) \leq \Theta^{\hat{s}}_{\mathrm{opt}} + \Delta$ holds. Thus, we have

$$\Theta^{\hat{s}_+}_{\mathrm{opt}} \leq \Theta^{\hat{s}}_{\mathrm{opt}} + \Delta. \tag{6.15}$$

By (6.9), (6.14) and (6.15), we obtain $R^{\hat{s}_+}(v) \geq R^{\hat{s}}(v)$.
(II): In this case, $\Theta^{\hat{s}_-}(v) = \Theta^{\hat{s}_-}(v, \hat{u})$ and $\Theta^{\hat{s}_-}(v, \hat{u}) = \Theta^{\hat{s}}(v, \hat{u})$ by (6.7) and (6.13). Thus, we have

$$\Theta^{\hat{s}_-}(v) = \Theta^{\hat{s}}(v). \tag{6.16}$$

By the optimality of $x^{\hat{s}_-}_{\mathrm{opt}}$ under $\hat{s}_-$, $\Theta^{\hat{s}_-}_{\mathrm{opt}} \leq \Theta^{\hat{s}_-}(x^{\hat{s}}_{\mathrm{opt}})$ holds. Here, we claim that $\Theta^{\hat{s}_-}(x^{\hat{s}}_{\mathrm{opt}}) \leq \Theta^{\hat{s}}_{\mathrm{opt}}$ holds, we thus have

$$\Theta^{\hat{s}_-}_{\mathrm{opt}} \leq \Theta^{\hat{s}}_{\mathrm{opt}}. \tag{6.17}$$

By (6.9), (6.16) and (6.17), we obtain $R^{\hat{s}_-}(v) \geq R^{\hat{s}}(v)$. □

Here, we have the following claim by Lemma 22.

Claim 25. *For a scenario $s \in \mathcal{S}$, function $\Theta^s(x)$ is unimodal in x when x moves along a path from a leaf to another leaf in T.*

For a given scenario $s \in \mathcal{S}$, by definition of (6.9) and Claim 25, function $R^s(x)$ is unimodal in x along a path from a leaf to another leaf in T. Thus, function $R_{\max}(x)$ is also unimodal in x since it is the upper envelope of unimodal functions by (6.10).

Lemma 26. *Along a path from a leaf to another leaf in T, function $R_{\max}(x)$ is unimodal in x.*

We also have the following claim by Lemma 23.

Claim 26. *For a scenario $s \in \mathcal{S}$ and a vertex $v \in V$, if $\hat{u} = \mathrm{argmax}\,\{\Theta^s(v, u) \mid u \in \delta(v)\}$ holds, there exists $x^s_{\mathrm{opt}} \in T(V(v, \hat{u}) \cup \{v\})$.*

Here, suppose that $\hat{s} = \mathrm{argmax}\{R^s(v) \mid s \in \mathcal{S}\}$ and $\hat{u} = \mathrm{argmax}\{\Theta^{\hat{s}}(v, u) \mid u \in \delta(v)\}$ hold for a vertex $v \in V$. We now show that there also exists

the minimax regret sink location x^* in $T(V(v,\hat{u}) \cup \{v\})$. Suppose otherwise: there exists x^* in $T(v,u)$ or on an edge vu (not including endpoints) for some $u \in \delta(v)$ with $u \neq \hat{u}$. By Claim 26, there exists $x^{\hat{s}}_{\mathrm{opt}}$ in $T(V(v,\hat{u}) \cup \{v\})$. Now, let us consider a path which goes through $x^{\hat{s}}_{\mathrm{opt}}$, v and x^* in this order. Then, by Claim 25, $\Theta^{\hat{s}}(x)$ is increasing in x when x moves along this path from $x^{\hat{s}}_{\mathrm{opt}}$ to x^*, which implies that $\Theta^{\hat{s}}(x^*) > \Theta^{\hat{s}}(v)$ holds. Thus, $R^{\hat{s}}(x^*) > R^{\hat{s}}(v)$ also holds by (6.9). We have $R_{\max}(x^*) \geq R^{\hat{s}}(x^*)$ by the maximality of $R_{\max}(x^*)$ and $R^{\hat{s}}(v) = R_{\max}(v)$ by definition of $\hat{s}$, thus $R_{\max}(x^*) > R_{\max}(v)$ holds, which contradicts the optimality of x^*. By the above discussion, we obtain the following lemma.

Lemma 27. *For a vertex $v \in V$, if $\hat{s} = \mathrm{argmax}\{R^s(v) \mid s \in \mathcal{S}\}$ and $\hat{u} = \mathrm{argmax}\{\Theta^{\hat{s}}(v,u) \mid u \in \delta(v)\}$ hold, there exists the minimax regret sink location $x^* \in T(V(v,\hat{u}) \cup \{v\})$.*

6.4.2 Algorithm

In this section, we present an $O(n^2 \log^2 n)$ time algorithm that computes $x^* \in T$ which minimizes function $R_{\max}(x)$.

We first show how to compute $R_{\max}(v)$ for a given vertex $v \in V$. Given a dominant scenario $s \in \mathcal{S}_{\mathrm{d}}(v)$, $\Theta^s(v)$ can be computed in $O(n \log n)$ time, and by Theorem 14, Θ^s_{opt} can be computed in $O(n \log n)$ time. Thus by (6.9), $R^s(v)$ can be computed in $O(n \log n)$ time. By Lemma 25, we only need to consider $n-1$ dominant scenarios for v, thus, $R_{\max}(v)$ can be computed by (6.10) in $O(n^2 \log n)$ time.

Lemma 28. *For a vertex $v \in V$, $R_{\max}(v)$ can be computed in $O(n^2 \log n)$ time.*

In order to find the minimax regret sink location $x^* \in T$, we apply an algorithm similar to the one presented at Section 6.3.2. The algorithm maintains a vertex set $U_i \subseteq V$ which induces a connected subgraph of T including x^*. At the beginning of the procedure, the algorithm sets $U_1 = V$, and at i-th iteration, it finds a centroid m_i of $T(U_i)$, computes $R_{\max}(m_i)$ in the above mentioned manner, and sets $U_{i+1} = U_i \cap (V(m_i,u_i) \cup \{v\})$ where $u_i = \mathrm{argmax}\{\Theta^{\hat{s}}(m_i,u) \mid u \in \delta(m_i)\}$ and $\hat{s} = \mathrm{argmax}\{R^s(m_i) \mid s \in \mathcal{S}_{\mathrm{d}}(m_i)\}$. Note that, by Lemma 27, $T(U_{i+1})$ contains x^* if $T(U_i)$ includes x^*. The algorithm iteratively performs the same procedure until $|U_l|$ becomes two where $l = O(\log n)$. Suppose that there eventually remain two vertices m_l and

$u_l \in U_l$. At this point, x^* exists on the edge $m_l u_l$. Then, x^* can be computed in a similar manner discussed at Claim 11. We have the following claim.

Claim 27. *Suppose that* $u_l = \text{argmax}\{\Theta^{\hat{s}_1}(m_l, v) \mid v \in \delta(m_l)\}$ *and* $m_l = \text{argmax}\{\Theta^{\hat{s}_2}(u_l, v) \mid v \in \delta(u_l)\}$ *hold for adjacent two vertices* m_l *and* u_l*, where* $\hat{s}_1$ *and* $\hat{s}_2$ *are worst case scenarios for* m_l *and* u_l*, respectively. Let* t^* *denote the solution to an equation* $R_{\max}(m_l) - td(m_l, u_l)\tau = R_{\max}(u_l) - (1-t)d(m_l, u_l)\tau$*. Then,*
(i) *if* $0 \leq t^* \leq 1$ *holds,* x^* *is a point dividing the edge* $m_l u_l$ *with the ratio of* t^* *to* $1 - t^*$*,*
(ii) *if* $t^* < 0$ *holds,* $x^* = m_l$ *holds, and*
(iii) *if* $t^* > 1$ *holds,* $x^* = u_l$ *holds.*

As mentioned above, the algorithm correctly outputs the minimax regret sink location x^* after $O(\log n)$ iterations. Thus, by Lemmas 27 and 28, we obtain the following theorem.

Theorem 15. *The minimax regret sink location problem in a dynamic tree network with uniform capacity can be solved in* $O(n^2 \log^2 n)$ *time.*

6.5 Conclusion

In Section 6.3, we proposed an $O(n \log n)$ time algorithm for the optimal 1-sink location problem in dynamic tree networks with uniform capacity. In Section 6.4, we proposed an $O(n^2 \log^2 n)$ time algorithm for the minimax regret 1-sink location problem in dynamic tree networks with uniform capacity, which uses the algorithm proposed in Section 6.3 as a subroutine.

Chapter 7

Conclusion

In this dissertation, we considered two problems arising from the viewpoint of evacuation planning, called space exploration problems under incomplete information and sink location problems in dynamic networks under incomplete information.

In Chapter 3, we considered the online exploration problems in cycles by two searchers and trees by p searchers. For each problem, we proposed an online algorithm whose optimality is guaranteed by the competitive ratio.

In Chapter 4, we considered the online exploration problem in simple polygons by a single searcher. For the problem, we proposed an online algorithm and showed an upper bound and a lower bound of competitive ratio. Also, we showed that the competitive ratio of the same algorithm can decrease if we restrict the class of object space to rectilinear simple polygons.

In Chapter 5, we considered the minimax regret sink location problems in dynamic path networks with uniform capacity. We treated two cases: the number of sink to be located is single or multiple. For both cases, we proposed algorithms which can solve the problems in polynomial time. Also, in this chapter, we considered the optimal k-sink location problem in a dynamic path network with general capacities and proved that the problem can be also solved in polynomial time.

In Chapter 6, we considered the minimax regret 1-sink location problem in dynamic tree networks with uniform capacity. For the problem, we proposed an algorithm which can solve the problem in polynomial time.

The above results are theoretically new and each of these has the academic value independently. In addition, our results give the mathematical foundation to construct the quantitative policy for the real evacuation planning.

We conclude this chapter by remarking future directions of research con-

cerning the online exploration problem and the minimax regret sink location problem in dynamic networks.

For the online exploration problem in undirected graphs, a number of interesting problems remain open, and we could consider many variations, e.g., in a geometric setting or using searchers with different features. We briefly mention the following two problems which are of particular interest. The first is to develop a better exploration algorithm for trees than greedy algorithms. Indeed, there is still a gap between upper bound $\frac{p+\lfloor \log p \rfloor}{1+\lfloor \log p \rfloor}$ and lower bound $\Omega(\log p / \log \log p)$ of [20]. The first step in this direction is to consider the case $p = 3$. In this case our upper bound is 2, and indeed this can be easily achieved by the following algorithm: one searcher leaves at the root and the exploration is done by the other two searchers; by implicitly regarding a tree as a cycle, one traces the tree in clockwise ordering while the other does in counterclockwise ordering until they meet. It is not difficult to see that the exploration time is upper bounded by $L + d_{\max}$. Since the cost of the optimal is lower bounded by $\max\{\frac{2L}{3}, 2d_{\max}\}$, the competitive ratio is bounded by 2. An instance of Fig. 3.5 shows that the competitive ratio of any greedy algorithm is at least 2 for $p = 3$, but in general only $\frac{5}{3}$ lower bound is known. So, a natural question is whether any cooperation of three searchers improves the upper bound of competitive ratio. Another problem is to develop an $O(p/\log p)$-competitive algorithm for general graphs. Consider the easier uniform length case. We apply the algorithm for tree exploration by implicitly constructing a spanning tree on the graph during the exploration. By (3.7), the exploration time can be upper bounded by $\frac{2(|V|-1+d\log p)}{\log p}$, where d is the length of a path from the root to the last vertex v_{last}. The optimal exploration time takes at least $\frac{2(|V|-1)}{p}$ time, which bounds the term $|V|$. However, to bound the term d, we need to introduce some depth-first search phase. For $p = 1$, we can use the algorithm by Duncan et al. [17] for our purpose, which shows how a tethered robot explores a graph with bounded diameter. (Indeed, Fleisher et al. [21] proposed to use the algorithm by Duncan et al. [17] for a similar purpose.) Developing an online graph exploration algorithm by p tethered robots would be a useful and challenging problem.

For the online exploration problem in polygons, as one of many variations of problem, we could consider the case with multiple searchers. In this problem, all searchers are initially at the same origin $o \in P$. The goal of the exploration is that each polygon vertex is visited by at least one searcher and that all searchers return to the origin o. We regard the time when the last

searcher comes back to the origin as the cost of the exploration. Note that our algorithm proposed in Chapter 4 can be easily adapted to the case with two searchers. For an offline exploration problem with k-searchers, Frederickson et al. [25] proposed an $(e + 1 - 1/k)$-approximation algorithm, where e is the approximation ratio of some 1-searcher algorithm. Their idea is splitting a tour given by some 1-searcher algorithm into k parts such that the cost of each part is equal, where the cost of a part is the length of the shortest tour from o which passes along the part. When $k = 2$, we can apply this idea to our algorithm as follows. First, choose similarly $e^* \in E^*$ satisfying (4.8) in Chapter 4. Then let one searcher go to $v^1_{e^*}$ and walk counterclockwise along the boundary of P, and let symmetrically the other go to $v^2_{e^*}$ and walk clockwise. When two searchers meet at a point on the boundary, two searchers come back together to o along the shortest path in the inside of P. In this case, we obtain an upper bound 1.719. However, when $k \geq 3$, the above-mentioned idea cannot be directly applied. So, it remains open.

For the minimax regret sink location problem in dynamic networks, there exist some open problems in discrete model. Throughout Part II, we considered all problems in continuous model. We can directly apply the proposed algorithms as mentioned in Part II to solve the problems also in discrete model without increasing the time complexity if the uniform capacity of edge is set to $c = 1$. However, the case of $c \geq 2$ is still left open. Recently, it has turned out that in discrete model with $c \geq 2$, there may exist a sink location for which any worst case scenario is not dominant, which implies that the algorithm has to consider more than $O(n)$ scenarios for a fixed sink (Guru Prakash and Prashanth Srikanthan and the authors of this paper, private communication, 2014). Fortunately, we can prove that the solution for continuous model can be regarded as the approximation for discrete model such that the difference between the approximate cost and the optimal cost is at most 1. In addition, we leave as an open problem to extend the solvable networks for the minimax regret sink location problem to more general dynamic networks (e.g., paths with general capacities, trees with general capacities, general undirected graphs with uniform capacity and so on). Indeed, under a fixed scenario, the algorithm by [33] can solve the optimal 1-sink location problem in dynamic tree networks with general capacities, we cannot simply apply this as a subroutine to solve the minimax regret version of the problem. For example, if Lemma 13 still holds in a dynamic tree network with general capacities, we can expect that an $O(n^2 \log^3 n)$ time algorithm will be achieved.

Bibliography

[1] S. Albers and M.R. Henzinger, "Exploring unknown environments", *SIAM J. Comput.*, 29(4), pp. 1164-1188, 2000.

[2] G. P. Arumugam, J. Augustine, M. J. Golin and P. Srikanthan, "A Polynomial Time Algorithm for Minimax-Regret Evacuation on a Dynamic Path", *CoRR abs/1404.5448*, arXiv:1404.5448.

[3] G. Ausiello, M. Demange, L. Laura and V. Paschos, "Algorithms for the on-line quota traveling salesman problem", *Inf. Process. Lett.*, 92(2), pp. 89-94, 2004.

[4] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie and M. Talamo, "Algorithms for the on-line travelling salesman", *Algorithmica*, 130, pp. 560-581, 2001.

[5] I. Averbakh and O. Berman, "$(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective", *Discrete Applied Mathematics*, 75(3), pp. 201-216, 1997.

[6] I. Averbakh and O. Berman, "Algorithms for the robust 1-center problem on a tree", *European Journal of Operational Research*, 123(2), pp. 292-302, 2000.

[7] B. Bhattacharya and T. Kameda, "A linear time algorithm for computing minmax regret 1-median on a tree", *Proc. COCOON 2012*, LNCS 7434, pp. 1-12, 2012.

[8] M. Blom, S. O. Krumke, W. E. de Paepe and L. Stougie, "The online TSP against fair adversaries", *INFORMS Journal on Computing*, 13(2), pp. 138-148, 2001.

[9] P. Brass, A. Gasparri, F. Cabrera-Mora and J. Xiao, "Multi-robot tree and graph exploration", *IEEE Trans. Robotics*, 27(4), pp. 707-717, 2009.

[10] G. S. Brodal, L. Georgiadis and I. Katriel, "An $O(n \log n)$ version of the Averbakh-Berman algorithm for the robust median of a tree", *Operations Research Letters*, 36(1), pp. 14-18, 2008.

[11] B. Chen and C. Lin, "Minmax-regret robust 1-median location on a tree", *Networks*, 31(2), pp. 93-103, 1998.

[12] E. Conde, "Minmax regret location-allocation problem on a network under uncertainty", *European Journal of Operational Research*, 179(3), pp. 1025-1039, 2007.

[13] E. Conde, "A note on the minmax regret centdian location on trees", *Operations Research Letters*, 36(2), pp. 271-275, 2008.

[14] X. Deng and C. H. Papadimitriou, "Exploring an unknown graph", *J. Graph Theory*, 32(3), pp. 265-297, 1990.

[15] W. Dinkelbach, "On nonlinear fractional programming", *Management Science*, 13(7), pp. 492-498, 1967.

[16] S. Dobrev, R. Královič and E. Markou, "Online graph exploration with advice", In *Proc. SIROCCO 2012* (LNCS 7355), pp.267-278, 2012.

[17] C. A. Duncan, S. G. Kobourov and V. S. A. Kumar, "Optimal constrained graph exploration", *ACM Trans. Algorithms*, 2(3), pp. 380-402, 2006.

[18] M. Dynia, M. Korzeniowski and C. Schindelhauer, "Power-aware collective tree exploration", In *Proc. ARCS 2006* (LNCS 3894), pp. 341-351, 2006.

[19] M. Dynia, J. Kutylowski, F. Meyer auf der Heide and C. Schindelhauer, "Smart robot teams exploring sparse trees", In *Proc. MFCS 2006* (LNCS 4162), pp. 327-338, 2006.

[20] M. Dynia, J. Lopuszański and C. Schindelhauer, "Why robots need maps", In *Proc. SIROCCO 2007* (LNCS 4474), pp. 41-50, 2007.

[21] R. Fleischer, T. Kamphans, R. Klein, E. Langetepe and G. Trippen, "Competitive online approximation of the optimal search ratio", *SIAM J. Comput.*, 38(3), pp. 881-898, 2008.

[22] R. Fleischer and G. Trippen, "Exploring an unknown graph efficiently", In *Proc. 13th ESA* (LNCS 3669), pp. 11-22, 2005.

[23] L. R. Ford Jr., D. R. Fulkerson, "Constructing maximal dynamic flows from static flows", *Operations Research*, 6, pp. 419-433, 1958.

[24] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, A. Pelc, "Collective tree exploration", *Net-works*, 48(3), pp. 166-177, 2006.

[25] G. N. Frederickson, M. S. Hecht and C. E. Kim, "Approximation algorithms for some routing problems", *SIAM J. Comput.*, 7, pp. 178-193, 1978.

[26] S. K. Ghosh and R. Klein, "Online algorithms for searching and exploration in the plane", *Computer Science Review*, 4(4), pp. 189-201, 2010.

[27] F. Hoffmann, C. Icking, R. Klein and K. Kriegel, "The polygon exploration problem", *SIAM J. Comput.*, 31(2), pp. 577-600, 2002.

[28] B. Hoppe and É. Tardos, "The Quickest Transshipment Problem", *Mathematics of Operations Research*, 25, pp. 36-62, 2000.

[29] B. Kalyanasundaram and K.R. Pruhs, "Constructing competitive tours from local information", *Theoretical Computer Science*, 130, pp. 125-138, 1994.

[30] N. Kamiyama, N. Katoh and A. Takizawa, "An efficient algorithm for evacuation problem in dynamic network flows with uniform arc capacity", *IEICE Transactions*, 89-D(8), pp. 2372-2379, 2006.

[31] A. N. C. Kang and D. A. Ault, "Some properties of a centroid of a free tree", *Information Processing Letters*, 4(1), pp. 18-20, 1975.

[32] P. Kouvelis and G. Yu, *Robust discrete optimization and its applications*, Kluwer Academic Publishers, Dordrecht, 1997.

[33] S. Mamada, T. Uno, K. Makino and S. Fujishige, "An $O(n \log^2 n)$ algorithm for the optimal sink location problem in dynamic tree networks", *Discrete Applied Mathematics*, 154(16), pp. 2387-2401, 2006.

[34] N. Megow, K. Mehlhorn and P. Schweitzer, "Online graph exploration: New results on old and new algorithms", In *Proc. 38th ICALP* (LNCS 6756), pp. 478-489, 2011.

[35] S. Miyazaki, N. Morimoto and Y. Okabe, "The online graph exploration problem on restricted graphs", *IEICE Trans. Inf. & Syst.*, E92-D(9), pp. 1620-1627, 2009.

[36] H. Nagamochi and K. Okada, “Approximating the minmax rooted-tree cover in a tree”, *Information Processing Letters*, 104, pp. 173-175, 2007.

[37] W. Ogryczak, “Conditional median as a robust solution concept for uncapacitated location problems”, *TOP*, 18(1), pp. 271-285, 2010.

[38] P. Panaite and A. Pelc, “Exploring unknown undirected graphs”, *J. Algorithms*, 33(2), pp. 281-295, 1999.

[39] J. Puerto, A. M. Rodríguez-Chía and A. Tamir, “Minimax regret single-facility ordered median location problems on networks”, *INFORMS Journal on Computing*, 21(1), pp. 77-87, 2009.

[40] S. Schaible and T. Ibaraki, “Fractional programming”, *European Journal of Operational Research*, 12, pp. 325-338, 1983.

[41] 赤木 徹也, 鯵坂 誠之, “高齢者の視覚探索特性に基づく環境の分かりやすさ”, 日本建築学会計画系論文集, 75(650), pp. 813-820, 2010.

[42] 秋月 有紀, 奥田 紫乃, 岩田 三千子, 田中 哮義, “立体角投射率による避難経路探索し易さの評価方法: 円滑な避難誘導のための視環境計画に関する研究その 2”, 日本建築学会環境系論文集, 77(674), pp. 231-239, 2012.

[43] 大畑 大志郎, 高井 伸雄, 鏡味 洋史, “釧路市中心市街地における津波避難施設配置の評価: マルチエージェントシステムを用いた津波からの避難シミュレーションその 2”, 日本建築学会計画系論文集, (612), pp. 87-91, 2007.

[44] 徳山 豪, “オンラインアルゴリズムとストリームアルゴリズム”, 共立出版, 2007.

[45] 野津田 宗聡, 岸本 達也, “地域避難施設の最適割当てと最適配置手法に関する研究: 地域避難施設の配置計画手法に関する研究その 1”, 日本建築学会計画系論文集, (589), pp. 115-122, 2005.

[46] 森山 修治, 長谷見 雄二, 小川 純子, 佐野 友紀, 神 忠久, 蛇石 貴宏, “大規模地下街における避難行動特性に関する実験研究”, 日本建築学会環境系論文集, 637(74), pp. 233-240, 2009.

List of Publications

[A] S. W. Cheng, Y. Higashikawa, N. Katoh, G. Ni, B. Su and Y. Xu, "Minimax regret 1-sink location problems in dynamic path networks", *Proc. The 10th Annual Conference on Theory and Applications of Models of Computation* (TAMC 2013), LNCS 7876, pp. 121-132, 2013.

[B] Y. Higashikawa, J. Augustine, S. W. Cheng, M. J. Golin, N. Katoh, G. Ni, B. Su and Y. Xu, "Minimax Regret 1-Sink Location Problem in Dynamic Path Networks", *Theoretical Computer Science*, DOI: 10.1016/j.tcs.2014.02.010, 2014.

[C] Y. Higashikawa, M. J. Golin and N. Katoh, "Minimax Regret Sink Location Problem in Dynamic Tree Networks with Uniform Capacity", *Proc. The 8th International Workshop on Algorithms and Computation* (WALCOM 2014), LNCS 8344, pp. 125-137, 2014.

[D] Y. Higashikawa, M. J. Golin and N. Katoh, "Multiple Sink Location Problems in Dynamic Path Networks", *Proc. The 10th International Conference on Algorithmic Aspects of Information and Management* (AAIM 2014), LNCS 8546, 2014.

[E] Y. Higashikawa, M. J. Golin and N. Katoh, "Improved Algorithm for the Minimum Time Sink Location Problem in Dynamic Path Networks", *Proc. The 2014 Fall National Conference of ORSJ*, 2014.

[F] Y. Higashikawa and N. Katoh, "Online Vertex Exploration Problems in a Simple Polygon", *IEICE Transactions on Information and Systems*, Vol.E96-D, No.3, pp.489-497, 2013.

[G] Y. Higashikawa, N. Katoh, S. Langerman and S. Tanigawa, "Online Graph Exploration Algorithms for Cycles and Trees by Multiple Searchers", *Journal of Combinatorial Optimization*, Volume 28(2), pp. 480-495, 2014.